ACRE: Agent Conversation Reasoning Engine

David Lillis
School of Computer Science and Informatics
University College Dublin
Email: david.lillis@ucd.ie
Rem W. Collier
School of Computer Science and Informatics
University College Dublin
Email: rem.collier@ucd.ie

***Abstract*—Within Multi Agent Systems, communication by means of Agent Communication Languages has a key role to play in the co-operation, co-ordination and knowledge-sharing between agents. Despite this, complex reasoning about agent messaging and specifically about conversations between agents, tends not to have widespread support amongst general-purpose agent programming languages.**

ACRE (Agent Communication Reasoning Engine) aims to complement the existing logical reasoning capabilities of agent programming languages with the capability of reasoning about complex interaction protocols in order to facilitate conversations between agents. This paper outlines the aims of the ACRE project and gives details of the functioning of a prototype implementation within the AFAPL2 agent programming language.

I. Introduction

Communication is a vital part of a Multi Agent System (MAS). Agents make use of communication in order to aid mutual cooperation towards the achievement of their individual or shared objectives. The sharing of knowledge, objectives and ideas amongst agents is facilitated by the use of Agent Communication Languages (ACLs). The importance of ACLs is reflected by the widespread support for them in agent programming languages and toolkits, many of which have ACL support built-in as core features.

In many MASs, communication takes place by way of individual messages without formal links between them. An alternative approach is to group related messages into conversations: "task-oriented, shared sequences of messages that they observe, in order to accomplish specific tasks, such as a negotiation or an auction" [1].

This paper presents the Agent Conversation Reasoning Engine (ACRE). The principal aim of the ACRE project is to integrate interaction protocols into the core of existing agent programming languages. This is done by augmenting their existing reasoning capabilities and support for inter-agent communication by adding the ability to track and reason about conversations. Currently at the stage of an initial prototype, ACRE has been integrated with the AFAPL2 Agent Programming Language [2], which runs on the Agent Factory platform [3]. The longer-term goals of ACRE include its use within other mainstream programming languages.

The principal aim of this paper is to outline the goals of the ACRE project and to present the integration of the prototype system into AFAPL2.

This paper is laid out as follows: Section II outlines some related work on agent interaction. Section III then provides an overview of the aims and scope of the ACRE project. Following this, details of the integration of ACRE into the Agent Factory framework are given in Section IV. The relationships between message performatives and agent goals are discussed in Section V, followed by an example of a simple one-shot auction implemented via ACRE in Section VI. Finally, Section VII outlines some conclusions along with ideas for future work.

II. Related Work

In the context of Agent Communication Languages, two standards have found widespread adoption. The Knowledge Query and Manipulation Language (KQML) was the firstly widely-adopted format for agent communication [4]. An alternative agent communication standard was later developed by the Foundation for Intelligent Physical Agents (FIPA). FIPA ACL utilises what it considers to be a minimal set of English verbs that are necessary for agent communication. These are used to define a set of performatives that can be used in ACL messages [5]. These performatives, along with their associated semantics, are defined in [6].

Recognising that one-off messages are limited in their power to be used in more complex interactions, FIPA also defined a set of interaction protocols [7]. These are designed to cover a set of common interactions such as one agent requesting information from another, an agent informing others of some event and auction protocols.

Support for either KQML or FIPA ACL communication is frequently included as a core feature in many agent tookits and frameworks, native support for interaction protocols is less common. The JADE toolkit provides specific implementations of a number of the FIPA interaction protocols [8]. It also provides a Finite State Machine (FSM) behaviour to allow interaction protocols to be defined. Jason includes native support for communicative acts, but does not provide specific tools for the development of agent conversations using interaction protocols. This is left to the agent programmer [9, p. 130]. A similar level of support is present within the Agent Factory framework [10].

There do exist a number of toolkits, however, that do include support for conversations. For example, the COOrdination

Language (COOL) uses FSMs to represent conversations [11]. Here, a conversation is always in some state, with messages causing transitions between conversation states. Jackal [12] and KaOS [13] are other examples of agent systems making use of FSMs to model communications amongst agents. Alternative representations of Interaction Protocols include Coloured Petri Nets [14] and Dooley Graphs [15].

III. ACRE Overview

ACRE is aimed at providing a comprehensive system for modelling, managing and reasoning about complex interactions using protocols and conversations. Here, we distinguish between a *protocols*' and *conversations*. A protocol is defined as a set of rules that dictate the format and ordering of messages that should be passed between agents that are involved in prolonged communication (beyond the passing of a single message). A conversation is defined as a single instance of multiple agents following a protocol in order to engage in communication. It is possible for two agents to engage in multiple conversations that follow the same protocol.

Such an aim can only be realised effectively if a number of features are already available. These include:

- **Protocol definitions understandable by agents:** Interaction protocols must be declared in a language that all agents must be able to understand and share. This also has the advantage that the protocol definition is separated from its implementation in the agent, thus providing a programmer with a greater understanding of the format the communication is expected to take. ACRE uses an XML representation of a finite state machine for this purpose.
- **Shared ontologies:** A shared vocabulary is essential to agents understanding each other's communications. A shared ontology defines concepts about which agents need to be capable of reasoning.
- **Plan repository:** With the two above features in place, an agent may reason about the sequence of messages being exchanged, as well as the content of those messages. This reasoning will typically result in an agent deciding to perform some action as a consequence of receiving certain communications. In this case, it is useful to have available a shareable repository of plans that agents may perform so that new capabilities may be learned from others.

The presence of these features aid greatly in the realisation of ACRE's aims. The principal aims are as follows:

- **External Monitoring of Interaction Protocols:** At its simplest level, conversation matching and recognition of interaction protocols allows for a relatively simple tool operating externally to any of the agents. This can intercept and read messages at the middleware level and is suitable for an open MAS in which agents communicate via FIPA ACL. This is a useful tool for debugging purposes, allowing developers to monitor communication to ensure that agents are following protocols correctly. This is particularly important where conversation management has been implemented in an ad-hoc way, with incoming and outgoing messages being treated independently and without a strong notion of conversations.
- **Internal Conversation Reasoning:** On receipt of a FIPA ACL message, it should be possible to identify the protocol being followed by means of the `protocol` parameter defined in the message (for the specification of the parameters available in a FIPA ACL message see [16]). Similarly, the initiator of a conversation should also set the `conversation-id` parameter, which is a unique identifier for a conversation. By referring to the the protocol identifier, an agent can make decisions about its response by consulting the protocol specification. Similarly, the conversation identifier may be matched against the stored history of ongoing conversations.
  ACRE aims to use this information to analyse the status of conversations and generate appropriate goals for the agent to successfully continue the conversation along the appropriate lines for the protocol that is specified. The use of goals follows [17]. Goals represent the motivations of the participants in a conversation. Thus the agents' engagement in a particular conversation is decoupled from the individual messages that are being exchanged, allowing greater flexibility in reasoning about their reactions and responses.
- **Organisation of Incoming Messages:** It is possible that an agent communicating with agents in another system may receive messages that do not specify their protocol and/or conversation identifier. In this case, it is useful for the agent to have access to definitions of the protocols in which it is capable of engaging so as to match these with incoming messages so as to categorise the messages.
- **Agent Code Verification:** The ultimate aim of ACRE is to facilitate the verification of certain aspects of agent code. In particular, given integration of conversation reasoning into a programming language, it should be possible to verify whether or not an agent is capable of engaging in a conversation following a particular protocol.

IV. Agent Factory

Agent Factory is an extensible, modular and open framework for the development of multi agent systems [3]. The primary agent programming language packaged with Agent Factory is AFAPL2 [2], although it also includes support for other agent programming languages such as ALPHA [18] and AgentSpeak [9].

This principal aim of this paper is to outline the integration of ACRE with AFAPL2. AFAPL2 is an agent programming language that was initially based on the Agent0 language, with notions of belief and commitment at its core [19]. Its capabilities have been augmented since, however, with the addition of such features as goal reasoning [20] and roles [10].

The existing goal-reasoning capabilities of AFAPL2 (outlined in [20]) required some extension in order to be usable for the purposes of ACRE.

AFAPL2 contains two types of activities (code that allows an agent to perform some task): actions and plans. An action is

a simple activity that is implemented by way of a single Java class, known as an actuator. Actions are designed to be used as primitive activities that can be grouped together to carry out more complex tasks. A plan is such a grouping, making use of plan operators (such as operators to carry out several actions in sequence or in parallel) to combine actions. Each activity has three components:

- A *precondition* that specifies the circumstances in which the activity may be executed. This is expressed in terms of beliefs that the agent must have when attempting to execute the activity.
- A *postcondition* that indicates the anticipated mental state on successful completion of the activity. This is expressed in terms of beliefs the agent will expect to have once the activity has completed.
- The *body* indicates how the activity can be carried out: for actions this is a Java class name whereas for plans this is the expression of how the actions are combined for a more complex activity.

In the existing implementation of goal-handling, goals are achieved by comparing them with the postconditions of the activities that the agent is capable of performing. Figure 1 shows an example definition of a plan designed to check whether a host (identified by an IP address contained in the `?ip_addr` variable) is responding to ping requests (the actual code implementing the plan is omitted). The precondition `BELIEF(true)` is always satisfied. The postcondition `BELIEF(pingStatus(?ip_addr,?status))` indicates that on successful execution of this plan, the agent will expect to have a belief about the status of the IP address that it attempted to check.

```
PLAN checkPingStatus(?ip_addr) {
   PRECONDITION BELIEF(true);
   POSTCONDITION BELIEF(pingStatus(?ip_addr,?status));
      ...
}
```

Fig. 1. AFAPL2 Plan Definition (plan body omitted)

`GOAL(pingStatus(192.168.1.1,?status))` indicates that the agent aims to have a belief about the status of the host with the IP address `192.168.1.1`. This interpretation of the goal would be contained in the relevant ontology. Here, `?status` is a variable (indicated by the `?` sigil) that can match against anything. Thus it is not a goal to bring about a particular status; rather just to find out what that status is.

An agent having this goal would identify the `checkPingStatus` plan to be a candidate plan for its achievement.. This is the case for two reasons. Firstly, its postcondition matches the goal, meaning that the agent will anticipate its goal being achieved by a successful execution of this plan and secondly because its precondition is satisfied by the current belief set of the agent (since an agent will always believe `true` to be true). In deciding on the appropriate course of action, the goal reasoning engine will identify all such candidate activities and execute one. In the event that no candidate activities can be found, the goal is dropped as unachievable.

A significant drawback with this method of reasoning is that if no activity is available that can directly result in a goal state being brought about, no further effort is made to achieve it. However, this does not necessarily mean that the agent is incapable of achieving its goal. In the event of an activity being identified whose postcondition is expected to satisfy the goal but whose precondition is not satisfied by the current state of the agent, the modified goal reasoning engine examines other activities to evaluate whether any are available that can satisfy that precondition. An example of this reasoning process is given in Section VI.

V. Mapping Performatives to Goals and Beliefs

In AFAPL2, the existing method of handing message receipts is simply to adopt a belief that the message has been received, leaving it as an exercise to the application programmer to deal with this event. One reason behind this method is that there is currently no support for messages to be linked into conversations. In contrast, ACRE can analyse the conversations and protocols about which the agent is aware and generate more appropriate goals and beliefs whenever messages are received and sent.

The goals or beliefs that are generated depend on the context within which a message is sent. For example, a `propose` message is used to indicate that the sender proposes to perform some action under certain conditions. There are, however, more than one reason why an agent may receive such a message. In one situation, the proposal is unsolicited (for example to initiate a FIPA Propose Interaction Protocol [21]). In this case, the message has no prior context and is unrelated to any previous experience of the recipient. By its nature, a `propose` message requires a response and so the recipient agent must evaluate the proposal and communicate whether or not it is willing to accept the proposal. As such, this situation will result in the adoption of a goal indicating that this type of evaluation should take place.

In contrast, a proposal may have been solicited by the recipient. The message may be matched to an existing conversation, either by means of an explicit conversation ID or by matching its content against that expected by the relevant protocol. By analysing this conversation, the agent can identify whether or not a call for proposals was previously sent out. In sending such a call, the agent will have been pursuing some other goal and so the adoption of an additional goal to handle the proposal is not desirable. Instead, a belief is adopted to indicate that the proposal has been received.

This approach also allows the agent to engage in separate but related conversations with different agents concurrently, as is shown in the example in Section VI.

Another example of the run-time conversation reasoning is on the receipt of an `accept-proposal` message. In this case, the treatment is different because of the future messages that the relevant protocol may or may not require to be sent in response. Under some protocols, an `accept-proposal` message is the final message in the conversation (e.g. the FIPA Propose Interaction Protocol [21] or the Vickrey Auction shown in Section VI). Here, a goal should be adopted merely

to perform the task that has been proposed and accepted. No further communication is required.

In other cases, such as within a FIPA Contract Net Interaction Protocol [22], the recipient of the `accept-proposal` message is required to communicate the result of performing the stated action back to the sender. In this case, the goal to be satisfied is twofold: firstly to perform the action and then communicate the result of this action to the sender. In reality, only one goal is necessary, as it is impossible to communicate the result of an action that has not been committed. This should be reflected in the preconditions of any activity that communicates the result of an action.

In the case of the sender of a message, it is not necessary to generate these goals. Here, the message is sent by the agent as a result of it having a goal that must be satisfied.

VI. Example: Vickrey Auction

In order to demonstrate how the ACRE system works, we use a Vickrey Auction Interaction Protocol. Figure 2 illustrates the protocol using Agent UML [23].

A Vickrey auction is a non-iterated auction, in that each bidder submits only a single bid, which is either accepted or rejected. It is also a sealed-bid auction, in that bids are communicated only to the auctioneer. In a Vickrey Auction, the winner of the auction is the bidder who submits the highest bid, though the ultimate price paid is equal to the second-highest bid.

In this example, one agent is assumed to desire that a task be performed by another agent and requests other agents to submit proposals for the performance of this task. This agent is referred to as the "Auctioneer". The auction is initiated by the Auctioneer sending a `cfp` message to a number of potential "Bidder" agents. Each bidder considers the call for proposals and decides whether or not to participate in the auction. Having done so, it indicates its decision to the Auctioneer either by submitting a bid (via a `propose` message) or by explicitly declining to do so (using a `refuse` message).

After receiving all of these responses, the Auctioneer must decide which is the winner of the auction and communicate its decision to each of the Bidders. This is done by sending a `accept-proposal` message to the successful bidder and a `reject-proposal` message to those that are unsuccessful.

A. ACRE Implementation Example

As noted in the above section, a Vickrey auction is typically initiated by an agent that wishes to have some task performed by another agent. This will generally be indicated by the agent adopting a goal to have the task performed.

In this example, we consider a MAS consisting of agents that are situated in a virtual grid world that contains items that the agents are required to collect. We begin the case study in a situation where one agent (which will become the Auctioneer agent) has discovered the location of an item and wishes to have it collected. This is reflected by the adoption of a goal, which is shown in Figure 4.

The addition of this goal to the agent's mental state will cause the goal reasoning engine to evaluate the options open to it to satisfy this goal. One option may be to execute a plan such as that defined in Figure 5. This is a plan that allows the agent to carry out the task (i.e. collect the referenced item) itself, without the need for engaging in conversation with other agents. However, it may alternatively be the case that the agent is not capable of performing the collection itself (if, for example, it is a coordinator of other agents). In such a scenario, it may be necessary to engage with other agents in order to find another that will be capable of (and willing to) carry out the task instead. The holding of an auction is one common way of solving such a problem.

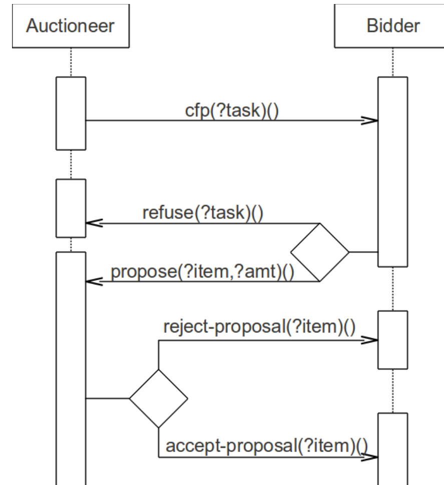

Fig. 2. AUML Diagram for a Vickrey-style auction

```
GOAL(performedTask(collected(item(20,25))))
```

Fig. 4. Initial goal to trigger a Vickery Auction

```
PLAN collectItem(?x,?y) {
   PRECONDITION BELIEF(true);
   POSTCONDITION
      BELIEF(performedTask(collected(item(?x,?y))));
      ...
}
```

Fig. 5. Plan Definition to allow an agent collect items (plan body omitted)

Sample code to implement an Auctioneer agent is presented in Figure 3. This include two plans used in the implementation of a Vickrey Auction. In addition to the two plans, the agent also includes an *AuctionModule*, which contains the code to reason about the bids that have been received and decide upon a winner. Two actuators are also present: one (addBid) to add a received bid to the AuctionModule and the other (endAuction) to trigger the ending of the auction and cause a winner to be decided upon. Finally, a perceptor is also present (auctionPerceptor) that monitors the state of the AuctionModule and adopts beliefs based thereon. These include beliefs about who the winners and losers of the auction are, following the end of the auction.

As outlined in Section IV, the goal reasoning engine firstly seeks an activity (either an action or a plan) whose postcon-

```
IMPORT com.agentfactory.afapl2.core.agent.FIPACore;
IMPORT agent.ACREAgent;

PLAN cfpTaskSolver(?task) {
   PRECONDITION  BELIEF(haveProposal(bidfor(?task,?bid),?agentID,?cid));
   POSTCONDITION BELIEF(performedTask(?task));

   BODY
      SEQ (
         FOREACH ( haveProposal(bidfor(?task,?amount),?agentID,?cid),
            addBid(?task,?agentID,?amount,?cid),
         ),
         endAuction,
         FOREACH ( BELIEF(status(?task,?agentID,winner)),
            accept-proposal(?agentID,?task)
         ),
         FOREACH ( BELIEF(status(?task,?agentID,loser)),
            reject-proposal(?agentID,?task)
         ),
         ADOPT(performedTask(?task))
      );
}

PLAN solicitProposals(?task) {
   PRECONDITION  BELIEF(neighbour(agentID(?aname,?aaddr)));
   POSTCONDITION BELIEF(haveProposal(bidfor(?task,?bid),agentID(?aname,?aaddr),?cid));

   BODY
      FOREACH ( BELIEF(neighbour(?agentID)),
         SEQ (
            cfp(?agentID,bidfor(?task)),
            OR (
               AWAIT(BELIEF(haveProposal(?bid,agentID(?aname,?aaddr),?cid))),
               AWAIT(BELIEF(haveRefusal(?task,agentID(?aname,?aaddr),?cid))),
               SEQ( DELAY(20), ADOPT(BELIEF(timeout(?agentID))))
            )
         )
      );
}

LOAD_MODULE AuctionModule module.AuctionModule;

PERCEPTOR auctionPerceptor {
   CLASS perceptor.AuctionPerceptor;
}

ACTION endAuction {
   CLASS actuator.EndAuctionActuator;
}

ACTION addBid( ?task, ?agentID, ?amount, ?cid ) {
   actuator.AddBidActuator;
}
```

Fig. 3. AFAPL2 Auctioneer Agent

dition satisfies the goal. In this example, the postcondition of the `cfpTaskSolver` plan will match the goal. This postcondition contains the variable `?task`, which is matched against the goal. This has the effect that the plan will be invoked with `collected(item(20,25))` set as the value for the `?task` variable.

However, this plan by itself will not be capable of bringing about successful achievement of the goal. This is because it also has a precondition that indicates that in order for the plan to succeed, the agent must already believe that it has received at least one other proposal from another agent to perform the task. As this is not the case, the goal reasoner must identify another activity that will bring about that precondition.

The `solicitProposals` plan has a postcondition that satisfies the precondition of `cfpTaskSolver` and is executable if the agent is aware of at least one neighbouring agent that it can invite to the auction. Thus the strategy the Auctioneer will employ will be to firstly execute

```
IMPORT agent.ACREAgent;

PLAN cfpProposal(?task, ?initiator, ?cid) {
   PRECONDITION  BELIEF(canBid(?task, ?amount, ?cid));
   POSTCONDITION BELIEF(respondedToCfp(bidfor(?task), ?initiator, ?cid));

   BODY
      propose(?initiator,bid(?task,?amount));
}

PLAN cfpRefusal(?task,?initiator,?cid) {
   PRECONDITION  BELIEF(noBid(?task,?cid));
   POSTCONDITION BELIEF(respondedToCfp(bidfor(?task),?initiator,?cid));

   BODY
      refuse(?initiator,bid(?task));
}

ACTION generateBid( ?task, ?cid ) {
   PRECONDITION  BELIEF(conversation(?cid,acre-vickrey));
   POSTCONDITION BELIEF(canBid(?task,?amount,?cid));

   CLASS is.lill.acre.actuator.GenerateBidActuator;
}
```

Fig. 6. AFAPL2 Bidder Agent

`solicitProposals` and then `cfpTaskSolver` in the expectation that the goal will be satisfied afterwards (by another agent performing the task).

The body of `solicitProposals` firstly considers all of the agents it has knowledge of (the `FOREACH` plan operator will execute the following code in parallel for every belief in the agent's belief set that matches `BELIEF(neighbour(?agentID))`, where `?agentID` can is bound in turn to the contents of each belief). For each of these agents it firstly sends a message to initiate the auction (the `cfp` action is part of the standard FIPACore agent that is imported at the top of the file). It then either waits until one of the following events has occurred: a) it believes it has received a proposal from the bidder, b) it believes that it has received a rejection from the bidder or c) some timeout period elapses, following which it adopts a belief to that effect. Once one of these things has occurred, the plan has completed.

It is important to note at this stage that the postcondition of `solicitProposals` may not be satisfied by its execution. The postcondition is designed to indicate the intended result of the plan, rather than enumerating all of its possible outcomes. In this case, the purpose of the plan is to solicit bids from other agents as part of an auction. Although it is possible that all agents could refuse to participate or fail to respond, it is not logical for an agent to issue a call for proposals in the hope that this will occur. From a goal-reasoning point of view, if no bids are received then the precondition of `cfpTaskSolver` is not satisfied and the goal is considered to be unsolvable. This is a logical outcome since the agent has no capability of performing the task itself and has failed to find another agent that is willing to do so on its behalf.

The beliefs about the receipt of a proposal or refusal are generated by ACRE reasoning about the conversations. This is an example of the situation presented in Section V where ACRE is aware that the proposal or refusal are in response to a call for proposals that was issued by the Auctioneer and so generates a belief rather than a goal.

If at least one bid is received then the precondition of `cfpTaskSolver` is satisfied and that plan may be executed. In this plan, the Auctioneer evaluates each of the proposals it has received and adds it to the AuctionModule that takes care of the decision-making with regard to the winner of the auction. Once all of the bids have been added, the auction can be ended. The auction perceptor will cause a set of beliefs about the auction to be adopted. These are used to send `accept-proposal` messages to the winner and `reject-proposal` messages to each of the losers of the auction.

In the context of the auctioneer, one advantage of this approach is that these plans are not limited to use within a Vickrey Auction. For example, a Contract Net Protocol [22] is also initiated by sending a `cfp` message and awaiting a response by means of either a `propose` or `refuse` message.

Figure 6 contains the AFAPL2 code for the Bidder agents. These agents must respond to the `cfp` message sent by the Auctioneer to initiate the auction. This is done by means of ACRE posting an appropriate goal for the agent's goal reasoner to solve. In this example, the goal is satisfied by the identical postconditions of the `cfpProposal` and `cfpRefusal` plans. However, there is no activity available with a postcondition that matches the precondition of `cfpRefusal`. On the other hand, `cfpProposal`'s precondition can be satisfied by executing the `generateBid` action (providing that its precondition that the conversation, represented by the variable `?cid` is of the type "acre-vickrey"). This is executed, followed by `cfpProposal`, assuming a belief that the agent is in a position to bid has been created.

The `generateBid` action may, however, cause the agent

to decline to make a bid (indicated by adopting a belief of the type `noBid`). This would mean that the precondition for the `cfpProposal` plan has not been satisfied and so it cannot be executed to satisfy the goal. At this point, the goal reasoner will re-evaluate the goal against the current belief set of the agent and, on finding the belief that the agent will not make a bid, now sees that the precondition of `cfpRefusal` is already satisfied by the current mental state of the agent. Thus this plan is called instead, causing a `refuse` message to be sent to the Auctioneer.

This example demonstrates one drawback of the use of postconditions in AFAPL2 to indicate the desired outcomes of activities. In this case, planning would be better facilitated by the express inclusion of `noBid` as a belief that will be adopted as an alternative outcome of the `generateBid` plan. As no express support is available for the enumeration of byproducts of activites (or the beliefs associated with a plan failing in its purpose).

No particular code is required to handle the response from the Auctioneer. In the event of the receipt of a `accept-proposal` message, this indicates that the Bidder is required to carry out some task that it has proposed to do, and so ACRE will adopt a goal to that effect. A `reject-proposal`, on the other hand, does not require any further action from the Bidder, so ACRE will merely adopt a belief to that effect that can be reasoned about by the agent.

VII. Conclusions and Future Work

This paper presents a prototype of the ACRE conversation reasoning system and specifically its integration into the AFAPL2 agent programming language. Although currently limited to AFAPL2, it is intended that ACRE will be used in conjunction with several other agent programming languages.

Although full integration with several languages is desirable, it may be necessary to adapt ACRE's workings to the specific needs and capabilities of particular languages. For example, not all languages support the use of preconditions and postconditions of activities to facilitate reasoning about them. On the other hand, support for ACL standards is widespread and so the grouping of messages into conversations is part of ACRE that is likely to be more widely applicable in its current form.

The availability of cross-platform communication tools such as ACRE, together with shared ontologies and protocol definitions can only aid interoperability between distinct agent platforms, toolkits and programming languages.

References

[1] Y. Labrou, "Standardizing agent communication," *Multi-Agents Systems and Applications (Advanced Course on Artificial Intelligence)*, pp. 74–97, 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=567252

[2] C. Muldoon, G. OHare, R. W. Collier, and M. OGrady, *Towards Pervasive Intelligence: Reflections on the Evolution of the Agent Factory Framework*. Boston, MA: Springer US, 2009, ch. 6, pp. 187–212. [Online]. Available: http://www.springerlink.com/content/g813865gq77731p1

[3] R. Collier, G. O'Hare, T. Lowen, and C. Rooney, "Beyond Prototyping in the Factory of Agents," in *Multi-Agent Systems and Application III: 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)*, Prague, Czech Republic, 2003.

[4] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," in *Proceedings of the Third International Conference on Information and Knowledge Management*, Gaithersburg, MD, 1994, pp. 456–463.

[5] S. Poslad, P. Buckle, and R. Hadingham, "The FIPA-OS Agent Platform: Open Source for Open Standards," in *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM2000)*, Manchester, 2000, p. 368.

[6] Foundation for Intelligent Physical Agents, "FIPA Communicative Act Library Specification," 2002. [Online]. Available: http://www.fipa.org/specs/fipa00037/

[7] ——, *FIPA Interaction Protocol Library Specification*, Std., 2000. [Online]. Available: http://www.fipa.org/specs/fipa00025/

[8] F. Bellifemine, G. Caire, T. Trucco, and G. Rimass, "Jade Programmer's Guide," 2007. [Online]. Available: http://jade.tilab.com/doc/programmersguide.pdf

[9] R. H. Bordini, J. F. Hübner, and M. J. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. Wiley-Interscience, 2007. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=AJHD4GkIQs0C&pgis=1

[10] R. Collier, R. Ross, and G. M. P. O'Hare, "A Role-Based Approach to Reuse in Agent-Oriented Programming," in *AAAI Fall Symposium on Roles, an Interdisciplinary Perspective (Roles 2005)*, Arlington, VA, USA, 2005.

[11] M. Barbuceanu and M. S. Fox, "COOL: A language for describing coordination in multi agent systems," in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995, pp. 17–24.

[12] S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, I. Soboroff, J. Mayfield, and A. Boughannam, "Jackal: a Java-based Tool for Agent Development," in *Working Papers of the AAAI-98 Workshop on Software Tools for Developing Agents*. AAAI Press, 1998.

[13] J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley, "KAoS: Toward an industrial-strength open agent architecture," *Software Agents*, pp. 375–418, 1997.

[14] R. S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng, "Modeling agent conversations with colored petri nets," in *In: Workshop on Specifying and Implementing Conversation Policies, Third International Conference on Autonomous Agents (Agents '99), Seattle*, 1999, pp. 59–66. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.113.6521

[15] H. Parunak, "Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis," in *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS)*, 1996.

[16] Foundation for Intelligent Physical Agents, "FIPA ACL Message Structure Specification," 2002. [Online]. Available: http://www.fipa.org/specs/fipa00061/

[17] L. Braubach and A. Pokahr, "Goal-Oriented Interaction Protocols," in *MATES '07: Proceedings of the 5th German Conference on Multiagent System Technologies*, vol. 4687. Leipzig, Germany: Springer, 2007, pp. 85–97.

[18] R. Collier, R. Ross, and G. M. P. O'Hare, "Realising Reusable Agent Behaviours with ALPHA," in *Proceedings of the 3rd German Conference on Multi-Agent System Technologies (MATES 05)*, Koblenz, Germany, 2005, pp. 210–215.

[19] Y. Shoham, "Agent0: An agent-oriented programming language and its interpreter," *Journal of Object-Oriented Programming*, vol. 8, no. 4, pp. 19–24, 1991.

[20] M. Dragone, D. Lillis, R. W. Collier, and G. M. P. O'Hare, "Practical Development of Hybrid Intelligent Agent Systems with SoSAA," in *Proceedings of the 20th Irish Conference on Artificial Intelligence and Cognitive Science*, Dublin, Ireland, August 2009.

[21] Foundation for Intelligent Physical Agents, "FIPA Propose Interaction Protocol Specification," 2002. [Online]. Available: http://www.fipa.org/specs/fipa00036

[22] Foundation For Intelligent Physical Agents, "FIPA Contract Net Interaction Protocol Specification," 2002. [Online]. Available: http://www.fipa.org/specs/fipa00029

[23] B. Bauer, J. Müller, and J. Odell, "Agent UML: A Formalism for Specifying Multiagent Software Systems," *Int. Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 3, pp. 207–230, 2001.