An Agent Communication Language for Negotiation in BDI Agents Environments

Myung Jin Lee†·Jin Sang Kim‡

ABSTRACT

Negotiation plays a fundamental role in human cooperative activities, allowing people to resolve conflicts that could interfere with cooperative behaviors. Negotiation in multi-agent systems is achieved through the exchange of messages in a shared agent communication language (ACL). We introduce a rational negotiation meta-language for autonomous, self-interested, and resource-bounded artificial BDI agents. We then propose a negotiation protocol for BDI agents with communicative acts based on their mental states.

1. Introduction

A language for inter-agent communication should allow agents to enlist the support of others to achieve goals, commit to the performance of actions for another agents, monitor their execution, report progress, success and failure, refuse task allocations, and acknowledge receipt of message. As such, it is essential that the functions being offered by the communication language be common across the language of intelligent agents. It so happens that there is such a language of speech acts, or more precisely, illocutionary acts which actions include requesting, promising, offering, acknowledging, proposing, accepting, etc.

A speech act is defined to be an action that a speaker performs in order to convey part of its mental state to the hearer that the act is directed to. This notion has been adopted in agent communication languages (ACL) such as Knowledge Query Manipulation Language (KQML) [1, 8] and Foundation for Intelligent Physical Agents (FIPA) ACL [2, 3], which prescribe the syntax and semantics of a collection of speech act-like messages, each of which is comprised of a performative, a content, and some additional parameters as the sender and receiver of the message.

KQML has the lack of semantics for communicative acts and so agent designers cannot be certain that the interpretation they are giving to a performative is in fact the same as the one some other designer intended it to have. FIPA ACL has several shortcomings. For example, the distinction among inform, tell, and assert is only relevant for an external observer. For an agent's subjective view they are equivalent. Also, an inform may be used answer queries which should, however, be done only by a reply.

In this paper, we consider three modalities: B for beliefs used to represent an agent's mental attitudes to the state of the environment, D for desires used to represent motivations of the agent, and I for intentions used to represent goals of the agent. We address rational communicative

† 정희현·아미스포럼대학교 인터넷비즈니스학과 교수
‡ 정희현·계명대학교 컴퓨터공학부 교수
논문접수: 2003년 7월 1일, 업무완료: 2003년 12월 30일
behaviors in autonomous, self-interested, resource-bounded artificial BDI agents that have what to communicate, to whom, and how. We then introduce a variant of KQML, FIPA ACL specification, and the negotiation meta-language as our negotiation language for BDI agents to resolve goal conflicts. Finally, we present a negotiation protocol for BDI agents with our negotiation language. When an agent receives a message from another agent, it interprets the message and generates a reply message based on its mental state.

2 Agent Communication Languages

2.1 KQML

KQML is intended as a general purpose communication language for the exchange of information and knowledge between software agents and it is conceptually a layered language [7]. It can be viewed as being divided into three layers: the content layer, the message layer, and the communication layer. The content layer is the actual content of the message in the programs own representation language. The message layer forms the core of the language. It determines the kinds of interactions one can have with a KQML-speaking agent. Finally, the communication level encodes a set of features to the message which describe the lower level communication parameters. KQML consists of logical combinations of the following five operators: \( \text{bel}(a, \phi) \) denoting that \( \phi \) is in the knowledge base of \( a \) and \( \text{know}(a, \phi) \), \( \text{want}(a, \phi) \), \( \text{intend}(a, \phi) \), standing for the fact that \( a \) knows \( \phi \), wants \( \phi \), and is committed to \( \phi \), respectively. Finally, there is an operator \( \text{process}(a, m) \) denoting that the message \( m \) will be processed by agent \( a \).

KQML is offered to the agent community as an extensible language with an open-ended set of performatives, whose meaning is independent of the propositional content language such as Prolog, first-order logic, and SQL. KQML, however, has yet to provide a precise semantics for these languages, as is customary with programming languages. Without one, agent designers cannot be certain that the interpretation they are giving to a performative is in fact the same as the one some other designer intended it to have. Moreover, the lack of semantics for communication acts leads to a number of confusions in the set of reserved performatives supplied.

2.2 FIPA ACL

FIPA ACL specification is based on speech act theory: messages are actions, or communicative acts, as they are intended to perform some action by virtue of being sent [4]. Every communicative act is described with both a narrative form and a formal semantics based on modal logic. The format of messages is similar to that of KQML, while their semantics are given by means of feasibility preconditions (FP) on the mental state of the sending agent that should hold prior to the dispatch of the message and rational effects (RE) that the sender can expect as a result of the dispatch. For example, inform act that agent \( i \) informs agent \( j \) that "it is true that it is raining today" can be represented as follows:

\[
\begin{align*}
\text{(inform)} \\
: \text{sender} \ i \\
: \text{receiver} \ j \\
: \text{content} \ "\text{weather(today, raining)}" \\
: \text{language} \ \text{Prolog}
\end{align*}
\]

Above inform act has the following underlying formal model:

\[
\langle i, \text{inform}(j, \varphi) \rangle
\]

\[
\begin{align*}
\text{FP} : & \quad B_{i} \varphi \land \lnot B_{i}(B_{j} \varphi \lor U_{j} \varphi) \\
\text{RE} : & \quad B_{j} \varphi
\end{align*}
\]

The semantics of inform act is that a sending agent \( i \) holds that proposition \( \varphi \) is true and \( i \) does not already believe that a receiver \( j \) has any knowledge of the truth of \( \varphi \), and, as the RE of the inform act, \( j \) believes that \( \varphi \) is true.

2.3 Negotiation meta-language

Wooldridge and Jennings [5, 6, 12] have explored two important computational problems in the use of logic-based languages for multi-agent negotiation: the success problem, i.e., given a particular negotiation history, has agreement been reached? and the guaranteed success problem, i.e., does a particular negotiation protocol guarantee that agreement will be reached?. They have considered three more complex languages for negotiation: classical propositional logic, a language for electronic commerce, and a negotiation meta-language.

In classical propositional logic, agents are negotiating over a domain that may be characterized in terms of a finite set of attributes, each of which may be either true or false. An outcome is thus an assignment of true or false to every attribute. In a language for electronic commerce, agents are trying to reach agreement on the values of a finite set of
negotiation issues, where each issue has a natural number value. They have also considered a negotiation meta-language which consists of illocutions: for example, request(i, j, φ) means a request from i to j for a proposal based on φ, where φ is a formula in classical propositional logic.

3 An Agent Communication Language for Negotiation

3.1 An ACL for BDI Agents

Our language is for negotiation between multiple agents over scarce resources. In this case, negotiation is achieved through the exchange of messages in a shared communication language. We assume that BDI agents meet the following requirements:

- Agents may interact asynchronously with more than one other agent at the same time;
- Agents are known to one another by their names, rather than their IP addresses; and
- The arguments of a message may affect the mental state of both the sender and the receiver.

We also consider that the B, D, and I are taken as meta-predicates. In order to represent these predicates, we take advantage of several characterizations in logic programming such as declarativity, unification, deduction rules, and meta-programming.

The actual exchange of messages is driven by the participating agents’ own needs, goals, or mental attitudes. We represent the set of beliefs as B, the set of desires as D, and the set of intentions as I. Each agent has a unique identifier and we denote the set of identifiers of the agents involved in negotiation as Agents. Assumed BDI agents are negotiating about the allocation of deficient resources, agents require the allocation of deficient resources to achieve their goals. We denote a set of goals as Goals and a set of the resources as Resources. In this case, we can define a communication language CL, communicative acts, for BDI agents as follows:

**Definition 1** Given a1, a2 ∈ Agents, g ∈ Goals, r ∈ Resources, and m ∈ B, D, or I, we define a CL:

- request(a1, a2, g, r) ∈ CL
- ask_if(a1, a2, m) ∈ CL
- inform(a1, a2, m) ∈ CL
- give(a1, a2, r) ∈ CL
- reject(a1, a2, g, r) ∈ CL

In definition 1, request(a1, a2, g, r) means that agent a1 requests deficient resources r from agent a2 to achieve its goal g where g is the reason why a1 needs r. ask_if(a1, a2, m) indicates that a1 asks a2 if m is true while inform(a1, a2, m) that a1 informs a2 that m is true.

3.2 A Negotiation Protocol for BDI Agents

Intelligent agents are software programs that use agent communication protocols to exchange information and to achieve their conflicting goals and resources allocation. The interaction protocols for intelligent agents are agent communication rules and based on speech-act language theory. They can be used as negotiation protocols which specify the messages that each agent is allowed to make.

In order to simplify protocol analysis, we assume that two BDI agents are involved in our negotiation protocol. The sequence of our negotiation protocol for BDI agents can be shown in (Figure 1) as a finite state diagram. In (Figure 1), S0~S7 represent different negotiation states during a negotiation process. S0 is the initial state and S7 is the terminal state in which an agreement or disagreement is reached. The process of negotiation starts when an agent generates a request message. This process continues until all the agents involved agree on a request or they cannot reach an agreement.

(Figure 1) A negotiation protocol for BDI agents

In our negotiation protocol, a sequence of simple negotiation message can be shown as follows:
State that starts a negotiation (S0). An agent knows that it wants to have any resources to achieve its goal. It sends a request message to another agent that has the resources. Negotiation processes start from this state.

State that receives a request (S1). An agent has received the request message, it interprets the request message, and generates a respond message. The message that can be generated in this state is a reject, a request, an alternative, or a give.

State that terminates the negotiation (S7). Two agents involved in the negotiation reach an agreement. The negotiation is terminated.

4. Experiments and Comparisons

4.1 Experiments

Our simple BDI agents’ negotiation mechanism is tested on InterProlog supporting Java 2 SDK version 1.3.1 and XSB Prolog version 2.4. InterProlog is a programming environment for XSB Prolog. It consists of a Java application front-end that communicates with a Prolog system running as a subprocess, using standard console redirection and TCP/IP sockets. It is implemented as a set of standard Java classes and Prolog predicates.

We consider three home improvement BDI agents with different objectives and resources. Agent a1 has the intention of hanging a picture, \( i(d(o(a1, hang\_picture)) \), and believes that it has in its possession a picture, a screw, a hammer, a hanger nail, and a screwdriver. It also believes that its name is a1 and agent a3 has a hanger. Agent a2 has the intention of hanging a mirror, \( i(d(o(a2, hang\_mirror)) \), and believes that it has a mirror and a nail. It also believes that its name is a2 and agent a1 has a screw, a hammer, and a screwdriver. Finally, agent a3 has the intention of hanging a clock, \( i(d(o(a3, hang\_clock)) \), and believes that it has a clock and a hanger. It also believes that its name is a3 and agent a1 has a hanger nail.

We construct BDI agents and allow them to negotiate with each other from the top-level window. NegotiationWindow class is responsible for creating this window, constructing the BDI agents, and starting a negotiation with the agents. <Table 1> illustrates the roles of the methods in NegotiationWindow class.

<table>
<thead>
<tr>
<th>Method names</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>negotiationWindow</td>
<td>Constructor to create the instance</td>
</tr>
<tr>
<td>constructWindowContents</td>
<td>Creates a 600 × 500 pane</td>
</tr>
<tr>
<td>constructMenu</td>
<td>Creates menu and menu, handles action events</td>
</tr>
<tr>
<td>createSimpleBDIAgent1</td>
<td>Creates agent 1’s window with Prolog engine</td>
</tr>
<tr>
<td>createSimpleBDIAgent2</td>
<td>Creates agent 2’s window with Prolog engine</td>
</tr>
<tr>
<td>createSimpleBDIAgent3</td>
<td>Creates agent 3’s window with Prolog engine</td>
</tr>
<tr>
<td>addItemToList</td>
<td>Adds items to menu</td>
</tr>
</tbody>
</table>

When we click Create simple BDI Agent 1 item in the top-level window, Agent1Window class displays Simple BDI agent 1 window with two panes: the top shows all outputs, stdout and stderr, from Prolog and the bottom is an editable text field which is sent to Prolog’s input, stdin, after hitting the Enter key. Agent1Window class then consults a routine of initializing agent a1 using sendAndFlush method in PrologEngine class. Agent2Window and Agent3Window classes perform similar tasks to Agent1Window class. PrologEngine class represents and gives access to a running Prolog process in background. Multiple instances correspond to multiple Prolog processes, outside the Java virtual machine. sendToFlush method in this class sends a string to Prolog’s input.

Let’s look at the initialization routine of a1 in the form of XSB Prolog. This routine consults negotiation mechanism routines for BDI agents and the knowledge base of a1. It also prepares a1 for communication with a2 and a3. There are many mechanisms of communication such as stream-oriented and message-oriented, but we use buffered, message-based communication mechanism using sockets: communication processes exchange messages that have well-defined boundaries, and use send_socket and socket3 to talk to each other where p/n represents predicate p has an arity of n.

On the other hand, the initialization routine of a2 consults negotiation mechanism routines for BDI agents and the knowledge base of a2. After it prepares a2 for communication with a1, a2 sends a message to a1, receives a confirmation message, and waits for a message from a1 to negotiate with a1. This means that we assume that a1 gives an
initial request, but if a2 does, we may have the same result.

When we click Start simple BDI agents negotiation item in the top-level window, negotiation among a1, a2, and a3 starts. Table 2 shows how these three agents can reach a mutual agreement state through a shared communication language. In Table 2, communication messages, ask-if(a1, a2, b(have(a2, nail))) and ask-if(a1, a3, b(have(a3, nail))) means that a1 asks a2 and a3 if they have a nail. a2 which receives an ask-if message from a1 checks its knowledge base, informs a1 that it has a nail, inform(a2, a1, b(have(a2, nail))). Now, a1 comes to know that a2 has a nail, requests a nail from a2, request(a1, a2, hang_picture, nail). This message means that a1 requests a nail from a2 to achieve its goal, hanging a picture.

Table 2 Communication messages in negotiation processes

<table>
<thead>
<tr>
<th>Direction</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1 → a2, a1 → a3</td>
<td>ask-if(a1, a2, b(have(a2, nail)))&lt;br&gt;ask-if(a1, a3, b(have(a3, nail)))</td>
</tr>
<tr>
<td>a2 → a1</td>
<td>inform(a2, a1, b(have(a2, nail)))</td>
</tr>
<tr>
<td>a1 → a2</td>
<td>request(a1, a2, hang_picture, nail)</td>
</tr>
<tr>
<td>a2 → a1</td>
<td>request(a2, a1, hang_mirror, hammer)</td>
</tr>
<tr>
<td>a1 → a2</td>
<td>alternativa(a1, a2, hang_mirror, screwdriver)</td>
</tr>
<tr>
<td>a2 → a1</td>
<td>request(a2, a1, hang_mirror, screwdriver)</td>
</tr>
<tr>
<td>a1 → a2</td>
<td>give(a1, a2, screwdriver)</td>
</tr>
<tr>
<td>a2 → a1</td>
<td>request(a2, a1, hang_mirror, screw)</td>
</tr>
<tr>
<td>a1 → a2</td>
<td>give(a1, a2, screw)</td>
</tr>
<tr>
<td>a2 → a1, a2 → a3</td>
<td>achieved_goal(a2, a1), achieved_goal(a2, a3)</td>
</tr>
<tr>
<td>a3 → a1</td>
<td>request(a3, a1, hang_clock, hang_nail)</td>
</tr>
<tr>
<td>a2 → a1, a3 → a2</td>
<td>give(a1, a3, hang_nail)</td>
</tr>
<tr>
<td>a3 → a1, a3 → a2</td>
<td>give(a1, a3, hang_nail)</td>
</tr>
<tr>
<td>a1 → a2</td>
<td>request(a1, a2, hang_picture, nail)</td>
</tr>
<tr>
<td>a2 → a1</td>
<td>give(a2, a1, nail)</td>
</tr>
<tr>
<td>a1 → a2, a1 → a3</td>
<td>achieved_goal(a1, a2), achieved_goal(a1, a3)</td>
</tr>
</tbody>
</table>

4.2 Comparisons

We defined the most basic inter-agent communication acts: inform, ask-if, request. In the case of request act, the KQML has achieve performative similar to request act. We consider a request act that a1 requests a2 to give the resources r. There is an important difference between KQML achieve, FIPA ACL request, and our request(a1, a2, g, r) act. KQML agent a2 does not know the reason why it sets the new value and FIPA agent a2 does not know the reason why a1 requests the resources r, while our BDI agent a2 knows the reason why a1 requests the resources r, i.e., because of the goal g. This fact helps an agent to reason about others' mental attitudes so that the agent may plan its goal more effectively.

The mental attitude m in definition 1 can be represented as follows: B(p), D(p), or I(p) where p is an atomic sentence. The use of inform for supplying new information to an agent is related to the data manipulation commands of SQL and Prolog. Assumed that an SQL database has belief, desire, and intention tables, called b_table, d_table, and i_table, respectively, an SQL INSERT of a new row <p> into the belief table or a Prolog assert(p) corresponds to sending an inform(a1, a2, B(p)) message. Sending inform(a1, a2, B(¬p)) with the negated sentence ¬p as content corresponds to an SQL DELETE of the respective row or a Prolog retract.

Table 3 shows these relationships between our communication acts, SQL, and Prolog.

Table 3 The relationships between our communication acts, SQL, and Prolog

<table>
<thead>
<tr>
<th>Our ACL</th>
<th>SQL</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>inform(a1, a2, B(p))</td>
<td>INSERT INTO b_table VALUES p</td>
<td>assert(p)</td>
</tr>
<tr>
<td>inform(a1, a2, B(p))</td>
<td>DELETE FROM b_table WHERE x = p</td>
<td>retract(p)</td>
</tr>
<tr>
<td>ask-if(a1, a2, B(p))</td>
<td>SELECT x FROM b_table WHERE x = p</td>
<td>?=p</td>
</tr>
<tr>
<td>ask-if(a1, a2, B(p))</td>
<td>n.a.</td>
<td>?=/=p</td>
</tr>
</tbody>
</table>

In this paper, when each agent interacts one another and notices inconsistency of its beliefs, we try to confine beliefs as much as possible, making each agent modify its beliefs. In Rao and Georgeff axiomatization [9–11], if an agent has an intention to do a particular action, the agent does the action. We also try to confine intentions to commit when the agent believes that it can do the action.

5. Conclusion and Future Research

In this paper, we have represented agents' beliefs, desires, and intentions for MAS in logic programming environments and introduced a negotiation mechanism for BDI agents. We have considered a MAS in which beliefs, desires, and intentions are taken as meta predicates and then introduced an ACL for BDI agents such as request, ask-if, and inform, described the relationships among our ACL, KQML, FIPA ACL, SQL, and Prolog. We finally introduced a negotiation protocol for BDI agents as a finite state diagram and defined a message generation function for BDI agents.

A number of issues raised in this paper require further investigation. We have considered the ACL containing some communication actions or performatives. Existing communi-
cation languages consider additional communication actions such as critique, withdraw, threaten, reward, persuade, and so on. We need to investigate whether some of these additional actions could be defined via negotiation protocols and have an assumption that different agents share the same communication language. However, this assumption is not essential. Indeed, translator agents could be defined, acting as mediators between agents with different communication languages. This may be possible by virtue of the metalogic features of the language.

References


