Flexible Communication of Agents based on FIPA-ACL

Jamshid Bagherzadeh [1]

Urmia University, Urmia, Iran

S. Arun-Kumar [2,3]

Indian Institute of Technology Delhi
New Delhi, India

Abstract

Communication in multi-agent systems is an important subject of the current research. In this paper, the syntax and semantics of a multi-agent programming language, called ECCS, are defined. We focus specially on the communication of agents. The main contribution of this paper is a new and flexible way of communication of agents. We finally work out a well known protocol as an example.

Key words: Multi-agent systems, agent communication languages, FIPA-ACL.

1 Introduction

A multi-agent system is a collection of agents, which have their individual goals (and perhaps some common goals), and they cooperate with each other (through an agent communication language), to fulfill their goals. In the last decade several new agent oriented languages and agent communication languages (ACLs), like KQML and FIPA-ACL, have been defined. Two of ACLs have got more attention in the literature: KQML [3] and FIPA-ACL [4]. The semantics of these ACLs have been defined in terms of conditions on the mental state of agents which is supposed to have beliefs, desires, intentions and so on.

We assume when an agent sends a message, it should satisfy some preconditions, and moreover after sending the message, it might update its local

[1] Email: jamshid@cse.iitd.ernet.in
[2] Email: sak@cse.iitd.ernet.in
[3] Partly supported by research grant F.26-1/2002-TS.V from MHRD, Govt. of India.

information to save necessary information which we call the effects of sending the message. On the other side, the receiver might update its local information upon receiving a message. In [1] we have defined a language, ECCS, for multi-agent programming, and defined the semantics of the language in terms of structural operational semantics [10]. One of the shortcomings of that semantics is the static nature of preconditions and effects of a message, i.e., the preconditions and effects of messages are fixed and the programmer cannot change them. This has two drawbacks, firstly, a programmer might not like the specified conditions and effects hold and he might like to define some other conditions and effects to be assumed during the sending or receiving a message. Secondly, the rationality of programs is low, in the sense that all the agents are treated in the same fashion, but the programmer might like to have different agents communicating in different ways. For example sending of a message to a trusted agent might have different effects than those of an untrusted agent. In this paper we have changed ECCS to capture this issue.

To do this, we have assumed a programmer will define a set of communicative actions which we call *performatives* where each performative has a precondition and a post-action. The precondition could be a formula and the post-action is a procedure which describes the changes in the information store of the agent after communication. For a particular performative, an agent either sends or receives the performative. For each of these circumstances the corresponding preconditions and effects need to be defined. The effect of sending a performative is diffrent from the effect of receiving the same performative. Any agent program has its own effects (perhaps different from other agents).

The literature contains several agent-oriented languages with a variety of communicative acts (eg. AGENT-0 [12], AgentSpeak(L) [11], GocGOLOG [7,5], 3APL [6], etc.). Even the acts themselves may be different, they are all beset with the same shortcommings as ECCS [1]. Although communication issues have been considered in these languages, they form only a small subset of communicative acts (CAs) mostly from KQML or FIPA-ACL.

Our attempt in this paper is to define a method which gives enough flexibility to the programmer to define new performatives and define their preconditions and effects on both sides of the communication. Thus the set of communication acts is not fixed and the new ones can be added.

We use CCS [8] and extend it with some primitive CAs from FIPA-ACL [4]. We define a structural operational semantics [10] of the language. This paper is organized as follows. In section 2 we show the structure of the the information store. Section 3 defines the syntax of agent programming language ECCS and the semantics of ECCS is defined in section 4. In section 5 we work out the FIPA-request protocol, and section 6 is the conclusion.

2 Information Store

Let $Ag = \{1, ..., n\}$ be a set of agents. Each agent has an Information Store (IS) which includes a set of beliefs and a set of goals. We use modal operators B_i and G_i to stand for *beliefs* and *goals* of agent $i \in Ag$ respectively. We have assumed a logic **KD45** for belief operator and a logic **KD** for goals. These axioms for belief operator are: (assume F and H are two formulas)

K: $\vdash B_i(F \Rightarrow H) \Rightarrow (B_iF \Rightarrow B_iH)$ **4**: $\vdash B_iF \Rightarrow B_iB_iF$

D: $\vdash B_iF \Rightarrow \neg B_i\neg F$ **5**: $\vdash \neg B_iF \Rightarrow B_i\neg B_iF$

In addition to above axioms we define following axioms which represent reasonable relations between beliefs and goals of an agent.

- $B_iG_iF \Leftrightarrow G_iF$
- $B_i\neg G_iF \Leftrightarrow \neg G_iF$

2.1 The Logic $PrBG_n$ [2]

Assume D is a domain representing the set of objects of the system, *Const* is a set of constants each of them representing an object of the domain D, *Var* is a set of variable symbols, and $\mathcal{P} = \{p, q, r, ...\}$ is a set of atomic predicates such that every atomic predicate has a predefined arity. A **term** t is either a constant or a variable (variables are universally quantified over the largest scope). Assume $i \in Ag$ is an agent, then formulas of $PrBG_n$ are defined as:

- **true** and **false** are formulas of $PrBG_n$;
- Any atomic predicate $p(t_1, \ldots, t_n) \in \mathcal{P}$ is in $PrBG_n$, where $t_1, \ldots, t_n$ are terms;
- If F and H are in $PrBG_n$ then so are $\neg F$, $F \vee H$, $F \wedge H$, B_iF , G_iF

We assume each agent (say i) has a finite **belief base**, denoted Ψ_{B_i}, and defined as:

$$\Psi_{B_i} \subseteq \{\omega \mid \text{where } \omega \text{ is a closed } PrBG_n \text{ formula}\}$$

A formula is closed if it does not have any free variables. Notice that belief or goal bases may have formulas with nested modalities. The semantics of the logic $PrBG_n$ is defined in a fairly standard fashion in [2]. We assume the set of beliefs is consistent, i.e., $\Psi_{B_i} \not\models$ **false**.

Any agent i has a finite set of goals called the goal base of i. The **goal base** of agent i, denoted as Ψ_{G_i}, is defined as:

$$\Psi_{G_i} \subseteq \{\omega \mid \text{where } \omega \text{ is a closed } PrBG_n \text{ formula}\}$$

Note that the goal base of an agent should be consistent, and it may not contain two inconsistent goals simultaneously. I.e., $\Psi_{G_i} \not\models$ **false**.

The satisfaction relation between pairs of form $\langle \Psi_i, \phi \rangle$ where $\Psi_i = (\Psi_{B_i}, \Psi_{G_i})$ is a (consistent) information store of agent i, and $\phi \in PrBG_n$, is defined as:

- $\Psi_i \models B_i\phi$ iff $\Psi_{B_i} \models \phi$
- $\Psi_i \models G_i\phi$ iff $\exists\omega \in \Psi_{G_i},\ \omega \models \phi$
- We define satisfaction relation for intention as: $\Psi_i \models I_i\phi$ iff $\Psi_i \models G_i\phi$ and $\Psi_i \not\models B_i\phi$.

First rule states that an agent believes some formula ϕ, if ϕ is a logical consequence of its belief base. The second rule states that agent i has ϕ as a goal if ϕ is a logical consequence of some formula $\omega \in \Psi_{G_i}$. The third rule states that an agent intends a formula ϕ if ϕ is a consequence of some goal and the agent does not believe ϕ.

In this paper we don't discuss the belief and goal revision in detail, as they have been discussed already in [2]. We have also given a sound and complete proof system in [2] for the propositional fragment of $PrBG_n$. We have assumed that the belief revision function defined there, works for both the belief and goal bases. We assume function *revise(*Ψ_{B_i}*,* ϕ*)* updates the belief base with ϕ, and *revise(*Ψ_{G_i}*,* ϕ*)* updates the goal base with ϕ. Moreover the operator $\sim$ is used for deleting formulas from the belief or the goal bases. To delete a formula ϕ from a belief base Ψ_{B_i}, we perform *revise(*Ψ_{B_i}*,* $\sim\phi$*)*. To remove a formula ϕ from the goal base we use *revise(*Ψ_{G_i}*,* $\sim\phi$*)* which removes the goals which imply ϕ.

3 Syntax of Agent Programming Language

The following BNF defines the programming language $\mathcal{L}$.

$$\begin{aligned}\pi ::=\ & 0 \mid \bar{c}(m).\pi \mid c(m).\pi \mid \mathit{update}(O_i\omega).\pi \mid \mathit{perform}(\mathit{Bact}).\pi \\ & \mid \mathit{query}(\omega).\pi \mid \mathit{observe}(l).\pi \mid \pi_1 + \pi_2 \mid A(\overrightarrow{t}).\pi\end{aligned} \tag{1}$$

The intuitive meaning of different constructs are as in CCS [8]. Operators $\bar{c}(m)$ and $c(m)$ are used for sending and receiving information between agents respectively, where c is an unidirectional communication channel between two agents (with an input port c and output port $\bar{c}$) and m is a message type or performative such as *inform(*ω*)* or *request(a)*. These performatives will be explained in the sequel. Operators *update(*$O_i\omega$*)* (where $O_i \in \{B_i, G_i\}$ and $\omega \in BG_n$) and *query(*ω*)* are used respectively for updating and querying the information store. *observe(l)* is used to observe the literal l from the environment, where $l = p$ or $l = \neg p$ and $p \in \mathcal{P}$ is an atomic predicate. *Bact* is an internal action (basic action) which is defined by the programmer, and *perform(Bact)* whould run *Bact*. Basic actions would be explained later.

The meaning of the composition operators is as usual. $a.\pi$ is a process that can perform action a and then become the process π, $\pi_1 + \pi_2$ is choice, which means either π_1 or π_2 will be executed. $A(\overrightarrow{t})$ is the call of a label (or procedure in imperative languages) with actual parameters $\overrightarrow{t}$.

Formally an ECCS agent is a tuple $M_i = \langle\ i,\ \Psi^0_{B_i},\ \Psi^0_{G_i},\ \mathit{Bacts},\ \mathit{Ch},\ \mathit{LB},$

Π, P_acts $\rangle$ where i is the identity of the agent, $\Psi^0_{B_i}$ is its initial belief base and $\Psi^0_{G_i}$ is initial goal base, $Bacts$ is a set of basic actions, and Ch is a list of channels which are used by i for communication. LB is a set of procedures which we call *labels*, and $\Pi \in \mathcal{L}$ is the main program. Finally P_acts is a set of performative acts to be used in communication.

A basic action is defined as $\langle\, b(\overrightarrow{x}) : \phi(\overrightarrow{x}), \psi(\overrightarrow{x})\, \rangle$, where b is the name of the action, $\overrightarrow{x}$ is a set of formal parameters and $pre(b(\overrightarrow{x})) = \phi(\overrightarrow{x}) \in BG_n$, and $post(b(\overrightarrow{x})) = \psi(\overrightarrow{x}) \in BG_n$ are its pre and postcondition respectively. A channel is defined as *c(s,r)* where c is the name of channel, s is the sender, and r is the receiver. Labels are defined as $A(\overrightarrow{x}) =_{def} \pi(\overrightarrow{x})$, where $\pi \in \mathcal{L}$ is a program of the grammar (1), and $\overrightarrow{x}$ is a set of formal parameters. A performative act (P_act) is similar to a basic action, with a precondition, but instead of the postcondition, it has a post-action which is a restricted program of the grammar (1) (without communication and labels). Any P_act shows the conditions to be satisfied before communication and the effects to be achieved after the action. P_acts would be discussed completely in the section 4.2.

Finally a multi-agent system is a parallel composition of various agents which may communicate together. Its syntax is defined as:

$$M = \; M_1 \mid M_2 | \; ... \; | M_n \qquad \text{where } M_i (i \in Ag) \text{ are agents.}$$

3.1 Performatives of FIPA-ACL

Some of the performatives of FIPA-ACL [4] are presented in table 1.

perf.	**syntax**	**semantics**
inform	$\langle s, inform(r,\omega)\rangle$	FP : $B_s\omega \wedge \neg B_s(Bif_r\ \omega \vee Uif_r\ \omega)$ RE: $B_r\omega$
request	$\langle s, request(r,a)\rangle$	FP : $(FP(a)[s\backslash r]) \wedge B_s Agent(r,a) \wedge \neg B_s I_r Done(a)$ RE : $Done(a)$
$\langle s, agree(r,a)\rangle =$		$\langle s, inform(r,\alpha)\rangle$ where $\alpha = I_s Done(a)$
$\langle s, refuse(r,a)\rangle =$		$\langle s, disconfirm(r,\alpha)\rangle$; $\langle s, inform(r,\beta)\rangle$
where $\alpha = Feasible(a)$, $\beta = \neg Done(a) \wedge \neg I_s Done(a)$		
$\langle s, failure(r,a)\rangle =$		$\langle s, inform(r,\alpha)\rangle$
$\alpha = (\exists e) Single(e) \wedge Done(e, Feasible(a) \wedge I_s Done(a)) \wedge \neg Done(a) \wedge \neg I_s Done(a)$		

Table 1
Some of the performatives of FIPA-ACL.

The first two performatives are primitive and the next three are composite performatives, which are defined based on the primitive ones. The syntax of each performative is of the form $\langle s, act(r,\alpha)\rangle$, where s is the sender, r is the receiver, act is the type of action, and α is the content of the message. The semantics of each performative consists of Feasibility Precondition (FP), which need to be satisfied before the act is performed, and Rational Effect

(RE), which is the effect expected by the sender after the act is performed.

In this table, $U_r\omega$ means that r is uncertain about ω. Although this operator is defined in the semantics of FIPA-ACL performatives, its semantics is vague, and we will ignore it in our framework. The problem of using the uncertainty operator appears when we combine it with beliefs and goals in the context of proof method defined in [2]. We believe that the proof method in this case becomes very difficult.

Moreover, in this table, $\mathit{Bif}_r(\omega) = B_r\omega \vee B_r\neg\omega$, $\mathit{Uif}_r(\omega) = U_r\omega \vee U_r\neg\omega$. $FP(a)[s\backslash r]$ denotes the part of the FPs of a which are mental attitudes of s, *Agent(r,a)* means that agent r can perform action a, *Done(a,ϕ)* means that action a is done (if $B_i\mathit{Done}(a,\phi)$) or intended to be done (if $I_i\mathit{Done}(a,\phi)$) and ϕ was true just before the performance of the action, *Done(a)=Done(a, true)*, *Single(e)* means action e is an atomic action and *Feasible(a)* means that preconditions of action a are satisfied. We will back to these in the section 4.2.

4 Semantics of ECCS

The semantics of ECCS is defined by Structural Operational Semantics (SOS) [10]. Many of the semantic rules are extensions of those for CCS [8]. If $\mathcal{P}$ is a set of atomic predicates then a literal, l, is either a predicate of $\mathcal{P}$ or its negation. Let c be a communication channel and m be a communication act or performative. Let also Act_{int} be the set of all internal actions. We assume Act_τ is a set of actions along with a distinguished invisible action τ. Formally:

$$Act_\tau = \{\tau\} \cup \{c(m), \bar{c}(m)\} \cup \{\mathit{observe}(l)\} \cup Act_{\mathit{int}}$$

The configuration of an agent includes those parts which might change during execution. For an agent, i, it is a tuple $\langle \Pi_i, \Psi_i, \theta_i, \mathit{did}(i,a)\rangle$, where Π_i is the continuation of the agent's main program, Ψ_i is its information store, which is defined as $\Psi_i = (\Psi_{B_i},\ \Psi_{G_i})$ representing the belief base and the goal base of i respectively, and θ_i is a variable-value binding which binds variables to values. *did(i,a)* specifies that action a was done by i in the previous transition.

The operational semantics of *update, query* and *basic actions* are defined in table 2. It is assumed that $O \in \{B, G\}$, $\omega \in PrBG_n$, and $\Psi'_i = (\Psi'_{B_i},\ \Psi'_{G_i})$ is the information store of agent i after revision. The operator *query(ω)* is used to query ω from the information store and bind the free variables. This operator is executed if there is a ground substitution η such that $\omega\theta_i\eta$ is satisfied by information store of agent i, otherwise it won't proceed. We assume η is a ground substitution such that $\mathit{Free}(\omega\theta_i) \subseteq \mathit{Dom}(\eta)$. Here $\mathit{Dom}(\eta)$ is the list of variables of η and $\mathit{Free}(\omega\theta_i)$ is the set of free variables of $\omega\theta_i$.

In the semantics of basic actions, $b(\overrightarrow{x})$ is a basic action, and $\overrightarrow{t}\theta_i$ is a ground list of terms and $(\overrightarrow{t}\theta_i/\overrightarrow{x})$ is a substitution of the parameters of $\overrightarrow{x}$ with the corresponding ground terms from $\overrightarrow{t}\theta_i$.

$$\frac{\Psi_i' = \mathit{revise}(\Psi_i, O_i\omega\theta_i)}{\langle \mathit{update}(O_i\omega).\Pi_i, \Psi_i, \theta_i, -\rangle \rightarrow \langle \Pi_i, \Psi_i', \theta_i, \mathit{did}(i, \mathit{update}(O_i\omega\theta_i))\rangle}$$
$$\frac{\Psi_i \models \omega\theta_i\eta}{\langle \mathit{query}(\omega).\Pi_i, \Psi_i, \theta_i, -\rangle \rightarrow \langle \Pi_i, \Psi_i, \theta_i\eta, \mathit{did}(i, \mathit{query}(\omega\theta_i\eta))\rangle}$$
$$\frac{\Psi_i \models \mathit{pre}(b(\overrightarrow{t}\theta_i/\overrightarrow{x}))\ \mathit{and}\ \Psi_i' = \mathit{revise}(\Psi_i, \mathit{post}(b(\overrightarrow{t}\theta_i/\overrightarrow{x})))}{\langle \mathit{perform}(b(\overrightarrow{t})).\Pi_i, \Psi_i, \theta_i, -\rangle \rightarrow \langle \Pi_i, \Psi_i', \theta_i, \mathit{did}(i, \mathit{perform}(b(\overrightarrow{t}\theta_i)))\rangle}$$

Table 2
Semantics of internal operations.

4.1 Semantics of Observe and Label

observe($l(\overrightarrow{t})$) where $l(\overrightarrow{t})$ is a literal, is used for updating the information of an agent by observing some atomic predicate from the environment. There are two purposes for observe: observing some new information from the environment or verifying the existing beliefs from the environment. In the following we assume η is a ground substitution such that $\mathit{Free}(\overrightarrow{t}\theta_i) \subseteq \mathit{Dom}(\eta)$, and *Env* denotes the environment and it contains the truth value of a set of related predicates.[4]

$$\frac{\exists\eta\ \mathit{Env} \models l(\overrightarrow{t}\theta_i\eta), \Psi_i' = \mathit{revise}(\Psi_i, B_i l(\overrightarrow{t}\theta_i\eta))}{\langle \mathit{observe}(l(\overrightarrow{t})).\Pi_i, \Psi_i, \theta_i, -\rangle \rightarrow \langle \Pi_i, \Psi_i', \theta_i\eta, \mathit{did}(i, \mathit{observe}(l(\overrightarrow{t}\theta_i\eta)))\rangle}$$

Note that if $l(\overrightarrow{t}\theta_i\eta)$ is already believed, then the revision will not change the set of beliefs. If $\overrightarrow{t}\theta_i$ is ground then the environment needs satisfy only $l(\overrightarrow{t}\theta_i)$. Finally if there is no substitution η such that $l(\overrightarrow{t}\theta_i\eta)$ is implied by the environment then we have the following rule:

$$\frac{\forall\eta : \mathit{Env} \not\models l(\overrightarrow{t}\theta_i\eta)}{\langle \mathit{observe}(l(\overrightarrow{t})).\Pi_i, \Psi_i, \theta_i, -\rangle \rightarrow \langle \Pi_i, \Psi_i', \theta_i, \mathit{did}(i, \tau)\rangle}$$

In this rule if $l(\overrightarrow{t}\theta_i)$ is ground then $\Psi'_{B_i} = \mathit{revise}(\Psi_{B_i}, \sim B_i l(\overrightarrow{t}\theta_i))$, i.e. $B_i l(\overrightarrow{t}\theta_i)$ will be deleted from the set of beliefs. But if $l(\overrightarrow{t}\theta_i)$ is not ground then $\Psi'_{B_i} = \Psi_{B_i}$. We assume at any time at most one substitution η may be observed for $\overrightarrow{t}\theta_i$. I.e., if *Env* has more than one possible substitution η, only one of them will be observed.

In the semantics of *label* execution, we will use the notion of **variant**. We say ρ' is a variant of ρ, if ρ' is obtained from ρ by renaming of free variables. Let $A(\overrightarrow{x}) =_{def} \pi(\overrightarrow{x})$ be a label definition, and $A(\overrightarrow{t})$ be a call of A with the

[4] Environment might be assumed as an agent, which is communicated by the *observe* command.

actual parameters $\overrightarrow{t}$. Let $\pi_v(\overrightarrow{x})$ be a variant of $\pi(\overrightarrow{x})$ where $\mathit{Free}(\pi_v(\overrightarrow{x})) \cap (\ \mathit{Free}(\Pi_i) \cup \mathit{Dom}(\theta_i)\ ) = \emptyset$. The semantics of $A(\overrightarrow{t})$ is defined as:

$$\frac{\langle \pi_v(\overrightarrow{t}\theta_i/\overrightarrow{x}), \Psi_i, \theta_i, \alpha\rangle \rightarrow \langle \pi', \Psi'_i, \theta'_i, did(i,a)\rangle}{\langle A(\overrightarrow{t}).\Pi_i, \Psi_i, \theta_i, \alpha\rangle \rightarrow \langle \pi'.\Pi_i, \Psi'_i, \theta'_i, did(i,a)\rangle}$$

4.2 Communication Operators

We have two operators for communication of agents: $c_{sr}(m)$ and $\bar{c}_{sr}(m)$ for receiving and sending information m through channel c_{sr}[5], respectively. We assume m is of the form *perf(α)*, where *perf* is a message type like *inform* or *request*, and α is the content of the message. Assume *perf* is a typical performative. To send a message, the sender must satisfy some preconditions. In addition there might be some revisions in the mental states of the agents involved in the communication. We don't fix any constant precondition and revisions of mental state in the semantics of the communication operators, instead, we let the programmer define these preconditions and appropriate revisions for any particular performative. The programmer is allowed to define a performative act (P_act), perf_{out} for any performative *perf*. This P_act has a precondition and a post-action which show the conditions to be satisfied before sending the *perf*, and the revisions after sending. Similarly the programmer can define a P_act, perf_{in} for any performative *perf*, to specify the corresponding revisions after receiving a message.

Formally these actions are defined as: $\mathit{perf}_{out}(r,\alpha) = \langle \omega(self, r, \alpha), \rho(self, r, \alpha)\rangle$ (for sending), and $\mathit{perf}_{in}(s,\ \alpha) = \langle -, \rho(self, s, \alpha)\rangle$ (for receiving), where r, α are formal parameters, and $\omega(self, r, \alpha)$ is a formula of $PrBG_n$ denotes the precondition of the performative *perf*. $\rho(self, r, \alpha)$ is a program of the following grammar (2) which shows the internal actions to be done by $self$ immediately after the communication. *self* is the owner of the action. Considering a procedure, instead of a formula in post-action, gives us more flexibility to act rationally. Using this method an agent may do different revisions after sending the same message to different agents. To avoid nonterminating procedures, we allow only the simple sequential programs with internal actions without communication acts and recursion. The formal syntax of this simplified language is:

$$\rho ::= 0 \mid update(O_i\omega).\rho \mid perform(b(\overrightarrow{t})).\rho \mid query(\omega).\rho \mid observe(p).\rho \mid \rho_1 + \rho_2 \tag{2}$$

We assume $\mathit{pre}(\mathit{perf}_{out}(r,\alpha)) = \omega$ and $\mathit{post}(\mathit{perf}_{out}(r,\alpha)) = \rho$. Note that perf_{in} does not have any precondition usually.

Example 4.1 Assume we want to define a performative called *inform* for the

[5] Remember that the channel c_{sr} has a sending port $\bar{c}_{sr}$ which is used by s and a receiving port c_{sr} which is used by r.

agent i. We might define a P_act *inform*$_{out}$ for agent i as:

inform$_{out}$ **(r,** α **)**:

```
pre  =  $\mathtt{B_{self}}\alpha \wedge \neg\mathtt{B_{self}}(\mathtt{B_r}\alpha \vee \mathtt{B_r}\neg\alpha)$
post =  update( $\mathtt{B_{self}B_rB_{self}}\alpha$ ).
        query( $\mathtt{B_{self}}$Trusts(r,self) ).
        update( $\mathtt{B_{self}B_r}\alpha \wedge \sim \mathtt{G_{self}B_r}\alpha$ )
```

Here *self* denotes the agent which includes the P_act, i.e., agent i in this example. We might also define a P_act *inform*$_{in}$ for agent i:

inform$_{in}$**(s,** α **)**

```
post= update( $\mathtt{B_{self}B_s}\alpha$ ).
      update( $\sim\mathtt{B_{self}G_s}$Done(a) ).
      query( Trustable(s) ). update( $\mathtt{B_{self}}\alpha$ )
```

where $a \in \{\ \bar{c}_{s(self)}(\text{inform}(\alpha)),\ \bar{c}_{s(self)}(\text{inform\_if}(\alpha))\ \}$

Preconditions of *inform*$_{out}$ are similar to those given in table 1. In its post-action we add $B_{self}B_rB_{self}\alpha$, and if *self* believes that r trusts *self*, then it will imply that r would believe α and deletes the goal for $B_r\alpha$ because *self* has believed $B_r\alpha$

Intuitively *self* after receiving *inform(α)* which is sent by s, believes that s believes α, because it supposes that s is sincere. In addition it will imply that s does not intend any more to do *inform(α)*, and it does not believe any more that s believes that *self* intends a (*self* might already intend that s should inform α). If *self* trusts s, then it believes α too.

4.2.1 Output action

First we define some operators used in the FIPA-ACL. We assume the operator *Feasible(a)*, where a is an atomic action or a choice of atomic actions, means that a is feasible or its precondition holds. A choice of actions is feasible if one of the choices is feasible. An atomic action is feasible if its precondition holds. Precondition of a communicative act, a, shown as *pre(a)* is defined by the corresponding *perf*$_{out}$ (if it is a send action), or *perf*$_{in}$ (if it is a receive action). Precondition of a basic action is defined in the definition of the basic action.

The feasibility of an action may be checked directly by checking its precondition, i.e., for an action a we may check *pre(a)*. This holds only when we know the exact precondition of a. In the case when we are informed that an action of another agent is feasible, we may not know the exact preconditions of the action. For example assume i is informed that j intends to make an action $\langle j, a\rangle$ feasible. In this case i may not have any idea about the precondition of a. However from the information that i has received it can imply that G_j *Feasible*$(\langle j,a\rangle)$.

As a result, we use two different (however semantically with the same meaning) notions to distinguish these cases. To check the exact precondition of an action we use the operator $pre(a)$ and to be aware of feasibility of an action, a, without knowing its exact precondition we use the predicate *Feasible(a)*.

The operator *Done(⟨i,a⟩, φ)*, similar to FIPA-ACL, describes a situation where the atomic action a has been done just in the previous step by i, and $\phi \in PrBG_n$ was true just before the performance of a. We should not store *Done(⟨i,a⟩, φ)* in the belief base of any agent, because it holds only in the state immediately after performance of a. Thus if we want to store *Done(⟨i,a⟩, φ)* in the belief base, it will become invalid just after the execution of another action.

We use three different symbols here: *HasDone(⟨i, a⟩)*, *Done(⟨i, a⟩)*, and *Do(⟨i, a⟩)*. *HasDone(⟨i, a⟩)* means that the action a has been done in some previous state. It is usually used inside a belief operator. *Done(⟨i, a⟩)* means a was just done. We do not use it inside any modal operator. This operator is usually used in the semantics of performatives. If it is needed to be stored in the IS, it should be transformed to *HasDone(⟨i, a⟩)* or *Do(⟨i, a⟩)*. *Do(⟨i, a⟩)* means that the action a may be done in the future. It is usually used inside a goal operator.

The operator *Agent(i, a)* describes that i is an agent which can do the action a. It is assumed that there is a global function *Agt* which any agent i can read globally, but write locally, i.e. only the actions which can be done by i might be updated by i. This function is defined as *Agt: Ag* $\times$ *Act* $\rightarrow$ *({true, false})* $\times\ \Phi$, which given an agent and an action, specifies whether the agent can perform the action or not, and what is the postcondition of the action. In this definition Φ is a set of $PrBG_n$ formulas, *Ag* is a set of agents and *Act* is a set of actions. To check B_i *Agent(j, a)*, we use the function *Agt* (with parameters j and a). Checking the nested formulas like B_iB_k*Agent(j,a)* also is possible using the function *Agt*, and it will reduce to check only *Agt(j,a)*. This is because, i knows that k has access to this function.

Let ρ, ρ' and ρ'' be processes of grammar (2), Ψ, Ψ', Ψ'' be information stores, θ, θ', θ'' be substitutions, and $\Pi \in \mathcal{L}$ be a main program. We define the *big-step* transition relation $\Longrightarrow$ as:

$$\frac{}{\langle \rho.\Pi, \Psi, \theta, - \rangle \overset{\epsilon}{\Longrightarrow} \langle \rho.\Pi, \Psi, \theta, - \rangle}$$

$$\frac{\langle \rho.\Pi, \Psi, \theta, - \rangle \xrightarrow{\tau} \langle \rho'.\Pi, \Psi', \theta', - \rangle \ , \ \ \langle \rho'.\Pi, \Psi', \theta', - \rangle \overset{\epsilon}{\Longrightarrow} \langle \rho''.\Pi, \Psi'', \theta'', - \rangle}{\langle \rho.\Pi, \Psi, \theta, - \rangle \overset{\epsilon}{\Longrightarrow} \langle \rho''.\Pi, \Psi'', \theta'', - \rangle}$$

The forth element of the configuration is not important in the transition $\Longrightarrow$ and we show it by $'-'$. Now we are ready to define the semantics of the output action. Let i and j be the sender and the receiver agents, c_{ij} be a channel from i to j, α be the contents of the message, ω=*pre(perf*$_{out}$*(j,*$\alpha\theta_i$*))*, and ρ= *post(perf*$_{out}$*(j,*$\alpha\theta_i$*))*, Semantics of message sending is defined as following:

$$\frac{\Psi_i \models \omega, \quad \langle \rho.\Pi_i, \Psi_i, \theta_i, - \rangle \overset{\epsilon}{\Rightarrow} \langle \Pi_i, \Psi_i', \theta_i', - \rangle}{\langle \bar{c}_{ij}(\mathit{perf}(\alpha)).\Pi_i, \Psi_i, \theta_i, - \rangle \ \rightarrow \langle \Pi_i, \Psi_i', \theta_i', \mathit{did}(i, \bar{c}_{ij}(\mathit{perf}(\alpha\theta_i))) \rangle}$$

This rule states that if ω is satisfied and the process $\rho.\Pi_i$ can perform some internal actions and change to Π_i then the process $\bar{c}_{ij}(\mathit{perf}(\alpha)).\Pi_i$ may perform the communication action and change to the process Π_i. The information store Ψ_i and the substitution θ_i are changed respectively. These changes are caused by the execution of the partial process ρ.

Table 3 represents the default definitions of $perf_{out}(j, \alpha))$ for some performative. We have written the post-actions of the performatives as a simple procedure, though a rational agent might have a complex procedure. Some

request$_{out}$ **(j,** a **)** **pre** = $\neg B_{self} G_j \ \mathit{Do}(\langle j, a \rangle) \wedge B_{self} \ \mathit{Agent(j,a)} \ \wedge G_{self} \ \mathit{Do}(\langle j, a \rangle)$ **post** = query($B_{self}\,\mathit{Trusts(j,self)}$). revise($B_{self} G_j \ \mathit{Do}(\langle j, a \rangle)$)
agree$_{out}$ **(j,** a **)** **pre** = $\beta \wedge \neg B_{self} \mathit{Bif}_j \beta$ **post** = revise($B_{self} B_j \beta$) /* $\beta = G_{self}$ Do($\langle self, a \rangle$) */
refuse$_{out}$ **(j,** a **)** **pre** = $\neg \ \mathit{pre(a)} \wedge \neg \mathit{Done}(self, a) \wedge \neg G_{self} \mathit{Do}(self, a) \wedge \neg B_{self} \mathit{Bif}_j \beta$ **post** = revise($B_{self} B_j \beta$) /* $\beta = \neg \ \mathit{Feasible(a)} \wedge \neg G_{self} \ \mathit{Do}(\langle self, a \rangle)$ */
failure$_{out}$ **(j,** a **)** **pre** = $\beta_1 \wedge \neg B_{self} \mathit{Bif}_j \beta_2$ **post** = revise($B_{self} B_j \beta_2$) /* $\beta_1 = \mathit{Done}(\langle self, - \rangle) \wedge \neg \ \mathit{Done}(\langle self, a \rangle) \wedge \neg G_{self} \ \mathit{Do}(\langle self, a \rangle)$ */ /* $\beta_2 = \neg \ \mathit{HasDone}(\langle self, a \rangle) \wedge \neg G_{self} \ \mathit{Do}(\langle self, a \rangle)$ */

Table 3
Default definitions of perf$_{out}$(j, α).

discussion is needed about table 3. In this discussion we usually use intention to mean as goal. In $\mathit{request}_{out}(j, a)$ the precondition is similar to its counterpart in table 1 without $FP(a)[self \backslash j]$, and moreover *self* has the goal that a be done by j. In its post-action, if *self* knows that j trusts *self*, it will imply that j will intend to do a. In contrast with table 1, here we do not consider $FP(a)[s \backslash j]$ for the sake of simplicity.

The preconditions of $\mathit{agree}_{out}(j, a)$ states that *self* intends to do a and does not believe that j is aware of that. So after performing the agree action, *self* believes that j also has believed that *self* intends to do a.

The preconditions of $\mathit{refuse}_{out}(j, a)$ expresses that a is not feasible, it is not done, *self* does not want to do a, and it does not believe that j is aware of these facts. So after sending the refuse action, *self* believes that j also has believed that a is not feasible, and it is not intended to be done by *self*.

The precondition of $\mathit{failure}_{out}(j, a)$ states that *self* believes that some action has been done, *Done(⟨ self, - ⟩)*, but this action is not a, ¬ *Done(⟨ self, a ⟩)*, and *self* does not want any more to do a. Moreover *self* does not believe that j is aware of these facts. So after sending the failure, *self* believes that j also has believed these facts. Note that we have used ¬*HasDone(⟨self,a⟩)* which states that a has not been done. The reason which we do not use ¬*Done(⟨self,a⟩)* was discussed already.

4.2.2 Input action

Assume $c_{ij}(perf(\alpha))$ denotes the receiving of a message, $\mathit{perf}_{in}(i,\alpha)$ is a P_act of agent j where i is the sender, $\rho = \mathit{post}(\ \mathit{perf}_{in}(i\theta_j, \alpha\theta_j\eta)\ )$, and η is a valuation of the free variables of $\alpha\theta_j$, which will be initialized during the communication. The unification η binds the free variables of $\mathit{perf}(\alpha\theta_j)$ and takes its values from the corresponding message which is sent by i. This will be clarified in the parallel composition rule. The semantics of receiving is defined as:

$$\frac{\theta'_j = \theta_j\eta, \quad \langle \rho.\Pi_j, \Psi_j, \theta'_j, - \rangle \overset{\epsilon}{\Rightarrow} \langle \Pi_j, \Psi'_j, \theta''_j, - \rangle}{\langle c_{ij}(perf(\alpha)).\Pi_j, \Psi_j, \theta_j, - \rangle \ \rightarrow \langle\ \Pi_j, \Psi'_j, \theta''_j, did(j, c_{ij}(perf(\alpha\theta_j\eta)))\ \rangle}$$

This rule states that first the substitution θ_j will be updated to include the substitution η. Then provided the partial process ρ capable of running in the new configuration, the process $c_{ij}(perf(\alpha)).\Pi_j$ may receive the message and become the process Π_j.

Table 4 represents the default post-action of the P_act, $perf_{in}(i, \alpha)$ for various performatives. Let us explain the meaning of the P_acts of the table 4.

request$_{in}$(**i**, a)	**pre** = -	
post = query($B_{self}\,\mathit{pre}(a)$). revise($G_{self}\,\mathit{Do}(\langle self, a\rangle)$) + query($\neg B_{self}\ \mathit{pre}(a)$). revise($\neg G_{self}\ \mathit{Do}(\langle self, a\rangle)$) }		
agree$_{in}$(**i**, a)	**pre** = -	**post** = revise($B_{self} G_i\ \mathit{Do}(\langle i, a\rangle)$)
refuse$_{in}$(**i**, a)	**pre** = -	**post** = revise($B_{self} B_i \beta \wedge B_{self}\beta$)
/* $\beta = \neg\ \mathit{Feasible}(\langle i, a\rangle) \wedge \neg G_i\ \mathit{Do}(\langle i, a\rangle)$ */		
failure$_{in}$(**i**, a)	**pre** = -	**post** = revise($B_{self} B_i\ \beta \wedge B_{self}\beta$)
/* $\beta = \neg\ \mathit{HasDone}(\langle self, a\rangle) \wedge\neg G_i\ \mathit{Do}(\langle i, a\rangle)$ */		

Table 4
Default post-action of the P_act $perf_{in}$.

The receiver of the *request(a)* will act according to $\mathit{request}_{in}(i, a)$. After receiving *request(a)*, if a is feasible then *self* will intend to do a, otherwise if a is not feasible then *self* will not intend to do a.

The receiver of *agree* will believe that i has intended to do a, according to the post-action of $agree_{in}(i,a)$. In the $refuse_{in}(i,a)$, *self* believes that a is not feasible and i does not have the goal to do it. The post-action of $failure_{in}(i,a)$ is similar to $refuse_{in}$.

4.3 Summation and Parallel composition

Semantics of summation $\Pi_1 + \Pi_2$ is defined as usual. Let $a \in Act_\tau$ then:

$$\frac{\langle \Pi_{i_1}, \Psi_i, \theta_i, - \rangle \rightarrow \langle \Pi'_{i_1}, \Psi'_i, \theta'_i, did(i,a) \rangle}{\langle \Pi_{i_1} + \Pi_{i_2}, \Psi_i, \theta_i, - \rangle \rightarrow \langle \Pi'_{i_1}, \Psi'_i, \theta'_i, did(i,a) \rangle}$$

$$\frac{\langle \Pi_{i_2}, \Psi_i, \theta_i, - \rangle \rightarrow \langle \Pi'_{i_2}, \Psi'_i, \theta'_i, did(i,a) \rangle}{\langle \Pi_{i_1} + \Pi_{i_2}, \Psi_i, \theta_i, - \rangle \rightarrow \langle \Pi'_{i_2}, \Psi'_i, \theta'_i, did(i,a) \rangle}$$

In the semantics of parallel composition there are two cases:
1: For any $l \in \{\tau\} \cup$ $literals$:

$$\frac{\langle \Pi_i, \Psi_i, \theta_i, - \rangle \rightarrow \langle \Pi'_i, \Psi'_i, \theta'_i, did(i,l) \rangle}{[...|\langle \Pi_i, \Psi_i, \theta_i, - \rangle|...] \rightarrow [...|\langle \Pi'_i, \Psi'_i, \theta'_i, did(i,l) \rangle|...]}$$

2: Let $\bar{c_{ij}}(perf(\alpha))$ be the message to be sent by i, and $c_{ij}(perf(\beta))$ be the message to be received by j. Then the communication of two agents i and j can be done if there is a most general unifier η such that $\alpha\theta_i = \beta\theta_j\eta$. The semantics of this communication is defined as:

$$\frac{\langle \Pi_i, \Psi_i, \theta_i, - \rangle \rightarrow \langle \Pi'_i, \Psi'_i, \theta'_i, did(i,\bar{\gamma}) \rangle \ , \ \langle \Pi_j, \Psi_j, \theta_j, - \rangle \rightarrow \langle \Pi'_j, \Psi'_j, \theta'_j, did(j,\gamma) \rangle}{\begin{array}{c}[...|\langle \Pi_i, \Psi_i, \theta_i, - \rangle|...|\langle \Pi_j, \Psi_j, \theta_j, - \rangle|...] \rightarrow \\ [...|\langle \Pi'_i, \Psi'_i, \theta'_i, did(i,\bar{\gamma}) \rangle|...|\langle \Pi'_j, \Psi'_j, \theta'_j, did(j,\gamma) \rangle|...]\end{array}}$$

where $\bar{\gamma} = \bar{c}_{ij}(perf(\alpha\theta_i))$ and $\gamma = c_{ij}(perf(\beta\theta_j\eta))$.

5 Example: FIPA request Protocol

We now explain FIPA request protocol in the context of our framework. This protocol allows one agent to request another agent to perform some action, and the receiving agent either performs the action or replies, in some way, that it cannot [4]. Figure 1 represents the implementation of this protocol in ECCS.

In this protocol the initiator starts with a request and the hearer replies either with a refuse or an agree and after agree it replies with an inform or failure. In the ECCS code we use some new notations instead of previously defined symbols. If $c = (i,j)$ is a channel from i to j, we use the notation $(i,j)!$ instead of $\bar{c}$ to represent that i sends a message to j. The notation (i,j) is used by j to represent that j receives a message from i. We use *Bel i p* to express B_ip, and *Goal i p* instead of G_ip.

```
Global function Agt = {(j,a) --> true }
Agent i:
Initial_beliefs ={ Bel i Trustable(j) }
Initial_goals ={ Goal i Do(<j,a>) }
Comm_channels ={ c=(i,j), d=(j,i) }
Main program:
  query(Goal i Do(<k,act>)).   /* act is bound to a, and k is bound to j */
  (i,k)!(request(act)).           /* sending a message to k */
  { (k,i)(agree(act)) +           /* receiving a message from k */
    (k,i)(refuse(act))
  }.
  { query(Bel i (Goal k (Do<k,act>))).
    { (k,i)(inform(Do(act))) +
      (k,i)(failure(act)) }
  }

Agent j:
Initial_beliefs ={Bel j Trusts(i,j), pre(a)} /*pre(a) are the preconditions of a*/
Initial_goals ={ }
Comm_channels ={ c=(i,j), d=(j,i) }
Main program:
  (i,j)(request(act)).   /* act is bound to a */
  { query(Goal j Do(<j,act>)).
      (j,i)!(agree(act)).
      perform(act).
      { (j,i)!(inform(post(<j,act>)))  +  /* either inform or failure will run
          (j,i)!(failure(<j,act>)) }        according to their preconditions */
  } +
  { query(not Goal j Do(<j,act>)).
    (j,i)!(refuse(<j,act>))
  }
```

Fig. 1. Implementation of the FIPA request protocol in ECCS.

Figure 2 shows a sample execution of this protocol between two agents i and j. The preconditions and post-actions of messages are applied from the previous tables. In the execution of protocols we have assumed that the addition of new beliefs or goals would result in the removal of the contradictory old beliefs or goals and their consequences. For example note that the last

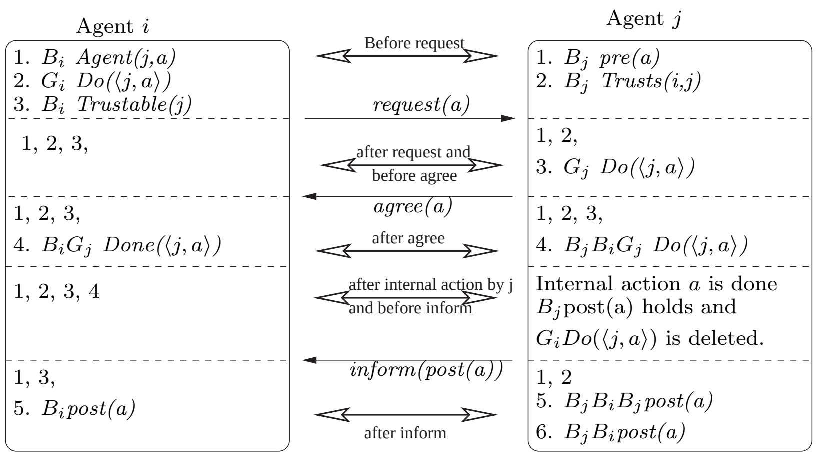

Fig. 2. Execution of the request protocol implemented in the figure 1.

state of agent i does not have formula 2 (because of B_i $post(\langle j,a\rangle)$ and formula 4 because of the post-action of $inform_{in}$. In the last state of j, formulas 3 and 4 are deleted because of the performance of a, and addition of 5 respectively.

6 Conclusion and Future Work

In this paper we have defined a language ECCS, with its syntax and semantics. We saw that how new communicative acts can be defined with appropriate operational semantics.

In [13] an operational semantics for agent communication languages has been proposed which suggests a similar work to be done in FIPA-ACL. One of the main points of this paper is the flexibility of defining various semantics for communicative acts and ability to define new performatives. However we have considered protocols of FIPA-ACL in our mind when defining some of the precondition and post-actions of performatives. Programmers can change the semantics when they want to use new protocols. In [9] an approach similar to this is suggested for communication.

This work extends the framework of [1] in various ways. The most importants are a new method of communication, and the use of a more expressive logic for information store taken from [2]. However because of space limit we have not put model checking algorithms here, but it can be defined in a similar fashion as [1]. In this case, we can simply check FIPA compliance properties as well as other properties of the agents. An issue for future work is to consider the complete set of FIPA-ACL performatives and protocols.

References

[1] J. Bagherzadeh and S. Arun-kumar. A multi-agent framework based on communication and concurrency. In *LNCS*, volume 3326. Springer-Verlag, 2004.

[2] J. Bagherzadeh and S. Arun-Kumar. Layered clausal resolution in the multi-modal logic of beliefs and goals. In *LNCS*, volume 3452, pages 544–559, 2005.

[3] T. Finin and et. al. KQML as an Agent Communication Language. In N. Adam and et. al., editors, *Proc. of CIKM'94*, pages 456–463, USA, 1994. ACM Press.

[4] Foundation for Intelligent Physical Agents(FIPA). Fipa2000 agent specification, http://www.fipa.org.

[5] G. De Giacomo, Y. Lespérance, and H. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[6] K. V. Hindriks and et. al. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[7] H. Levesque and et. al. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.

[8] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[9] J. Pitt and E. H. Mamdani. A protocol-based semantics for an agent communication language. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 486–491, 1999.

[10] G. D. Plotkin. A structural approach to operational semantics. Technical report, DAIMI, FN 19, Deptt. Comp. Sci., Univ. of Aarhus, Denmark, 1981.

[11] A. S. Rao. AgentSpeak(L): BDI Agents speak out in a logical computable language. In W. Van de Velde and J. Perram, editors, *Proc. of MAAMAW'96*, number 1038 in LNAI, pages 42–55, The Netherlands, 1996. Springer-Verlag.

[12] Y. Shoham. Agent-oriented programming. *Artif. Intell.*, 60(1):51–92, 1993.

[13] R. M. van Eijk and et. al. Operational semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 80–95. Springer-Verlag, 2000.