Flexible Communication of Agents based on FIPA-ACL

Jamshid Bagherzadeh [1] S. Arun-Kumar [2]

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 is 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 describe a logic to specify desirable properties of agents. This can be used to verify the correctness of agents. We finally work out a well known protocol as an example.

Key words: Multi-agent systems, agent communication languages, CCS, 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 (AOLs) 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 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,

[1] Email: jamshid@cse.iitd.ernet.in
[2] Email: sak@cse.iitd.ernet.in

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 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.

- $B_iG_i\phi \Leftrightarrow G_i\phi$
- $B_i\neg G_i\phi \Leftrightarrow \neg G_i\phi$

2.1 The Logic BG_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 BG_n are defined as:

- **true** and **false** are formulas of BG_n;
- Any atomic predicate $p(t_1, \ldots, t_n) \in \mathcal{P}$ is in BG_n, where $t_1, \ldots, t_n$ are terms;
- If F and H are in BG_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:

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

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 BG_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:

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

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 $\phi \in BG_n$, 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 BG_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 $\mathcal{L}$.

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

The intuitive meaning of different constructs are as in CCS [8]. Operators $\bar{c}(\alpha)$ and $c(\alpha)$ 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 α 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 $M_i = \langle\ i,\ \Psi^0_{B_i},\ \Psi^0_{G_i},\ \mathit{Bacts},\ \mathit{Ch},\ \mathit{LB},\ \Pi,\ \mathit{P\_acts}\ \rangle$ 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 4.1.

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

$M = \ M_1 \mid M_2 \mid ... \mid M_n$ where $M_i (i \in Ag)$ 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 : $Done(a)$
$\langle s, agree(r, < s, act >, \phi)\rangle$ =	$\langle s, inform(r, \alpha)\rangle$	where $\alpha = I_s Done(< s, act >, \phi)$
$\langle s, refuse(r, < s, act >, \phi)\rangle$=	$\langle s, disconfirm(r, \alpha)\rangle$; $\langle s, inform(r, \beta)\rangle$	
where $\alpha = Feasible(< s, act >)$, $\beta = \phi \wedge \neg Done(< s, act >) \wedge \neg I_s Done(< s, act >)$		
$\langle s, failure(r, a, \phi)\rangle = \langle s, inform(r, \alpha)\rangle$		
$\alpha = (\exists e)\, Single(e) \wedge Done(e, Feasible(a) \wedge I_s Done(a)) \wedge \phi \wedge \neg Done(a) \wedge \neg I_s Done(a)$		

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 $< s, act(r, \alpha) >$, 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, $Bif_r(\omega) = B_r\omega \vee B_r\neg\omega$, $Uif_r(\omega) = U_r\omega \vee U_r\neg\omega$. $FP(a)[s\backslash r]$ 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 $B_iDone(a,\phi)$) or intended to be done (if $I_iDone(a,\phi)$) 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 4.1.

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]. Let $\mathcal{P}$ be a set of atomic predicates, $p \in \mathcal{P}$ be an atomic predicate, and $l = p$ or $l = \neg p$ be a literal.

$$Act_\tau = \{\tau\} \cup \{c(\alpha), \bar{c}(\alpha) \mid \alpha \textit{ is a communicative act}\} \cup \{observe(l)\}$$

The configuration of an agent includes those parts which might change during execution. For an agent, i, it is a tuple $\langle \Pi_i, \Psi_i, \theta_i, did(i,a)\rangle$, 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 $O \in \{B, G\}$, $\omega \in BG_n$, and $\Psi'_i = (\Psi'_{B_i},\ \Psi'_{G_i})$ is the

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

Table 2
Semantics of internal operations.

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 $Free(\omega\theta_i) \subseteq Dom(\eta)$. Here $Dom(\eta)$ is the list of variables of η and $Free(\omega\theta_i)$ 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 $\overrightarrow{t}\theta_i$.

4.0.1 Observe:
$observe(l(\overrightarrow{t}))$ where l 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 $Free(\overrightarrow{t}\theta_i) \subseteq Dom(\eta)$, and Env denotes the environment and it contains the truth value of a set of related predicates. [3]

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

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:

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

In this rule if $l(\overrightarrow{t}\theta_i)$ is ground then $\Psi'_{B_i} = revise(\Psi_{B_i}, \sim B_i l(\overrightarrow{t}\theta_i))$, 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.

4.0.2 Label:
Let $A(\overrightarrow{x}) =_{def} \pi(\overrightarrow{x})$ be an agent name, and $\overrightarrow{t}$ a list of terms. The semantics of $A(\overrightarrow{t})$ is defined as:

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

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

4.1 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} [4], 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: $perf_{out}(r,\alpha) = \langle \omega(self, r, \alpha), \rho(self, r, \alpha) \rangle$ (for sending), and $perf_{in}(s, \alpha) = \langle -, \rho(self, s, \alpha) \rangle$ (for receiving), where r, α are formal parameters, and $\omega(self, r, \alpha)$ is a formula of BG_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:

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

The language obtained from this grammar is a subset of the language obtained from the grammar (1). We assume $pre(perf_{out}(r, \alpha)) = \omega$ and $post(perf_{out}(r, \alpha)) = \rho$. Note that $perf_{in}$ does not have any precondition usually.

Example 4.1 Assume we want to define a performative called *inform* in our framework. We might define a P_act $inform_{out}$ for agent i as:

inform$_{out}$ (**r,** α):
pre = $\mathtt{B}_{\mathtt{self}}\alpha \wedge \neg \mathtt{B}_{\mathtt{self}}\mathtt{Bif}_{\mathtt{r}}\alpha$
post = update($\mathtt{B}_{\mathtt{self}}\mathtt{B}_{\mathtt{r}}\mathtt{B}_{\mathtt{self}}\alpha$).
query($\mathtt{B}_{\mathtt{self}}$Trusts(r,self)).

[4] 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.

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 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)$\wedge \sim \mathtt{B_{self}B_sG_{self}}$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.

4.1.1 Output action

Before defining the semantics of the output action, we define some related concepts. We define *Done($\langle i,\rho \rangle$, ϕ)* to describe a situation where the expression ρ is done in the previous step by i, and $\phi \in BG_n$ was true just before the performance of ρ. We assume ρ is a program of the grammar 2. *Done($< i, \rho >, \phi$)* can be checked by looking at the previous actions of i which are kept in the configuration of i.

On the other hand an agent, say i, can not check the previous actions of another agent, say j. Thus *Done($\langle j, \rho\rangle$, ϕ)* should be stored explicitly in the IS of i. When *Done($\langle i, \rho\rangle$, ϕ)* comes in the context of a *goal* operator, (like G_i *Done($\langle i, \rho\rangle$, ϕ)* or G_i *Done($\langle j, \rho\rangle$, ϕ)*), it is manipulated as a predicate which should be explicitly kept in the information store. In general we have the following cases:

- B_i *Done($\langle i, \rho \rangle$, ϕ)* $\Leftrightarrow$ *Done($\langle i, \rho\rangle$,ϕ)* /* if ρ is really done by i */
- B_i *Done($\langle j, \rho\rangle$, ϕ)* $\wedge\ i \neq j \Leftrightarrow \Psi_i \models B_i$ *Done($\langle j, \rho\rangle$, ϕ)*
- G_i *Done($\langle j, \rho\rangle$, ϕ)* $\Leftrightarrow \Psi_i \models G_i$ *Done($\langle j, \rho\rangle$, ϕ)*

The next operator which is used in this section is *Feasible(ρ)*[5] , which means that ρ is feasible, and its preconditions hold. A sequence of actions $\rho = a.\rho'$ is feasible in the configuration C if the first action, a, is feasible in the configuration C, and in the next configuration, C', obtained from the effects of a on C, the sequence ρ' is feasible. This can be done inductively by applying

[5] Here ρ might have communication action also.

the effects of actions in the configurations. A choice of actions is feasible if one of the choices is feasible. If ρ is a communicative act then its preconditions is defined by the corresponding P_act, otherwise if it is an internal (or basic) action, then its precondition is defined in the corresponding basic action. In general we have the following cases:

- $\Psi_i \models B_i$ *Feasible(a) iff*
  $\Psi_i \models$ *pre(a)* if a is an internal or communicative action of i
  $\Psi_{B_i} \models$ *Feasible(a)* if a is an internal action of $j \neq i$
  $\Psi_{B_i} \models B_j$ *pre(a)* if a is a communicative act of $j \neq i$
- $\Psi_i \models G_i$ *Feasible(a) iff*
  $\Psi_{G_i} \models$ *pre(a)* if a is an internal or comunicative act of i
  $\Psi_{G_i} \models$ *pre(a)* if a is a comunicative act of $j \neq i$

It is not possible for i to make feasible an action a, if a is an internal action of $j \neq i$, because i does not know the preconditions of internal actions of j.

The next operator is *Agent(i, a)* which denotes that i is an agent which can do the action a. It is assumed that there is a global function *Agt* where 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*}, which given an agent and an action, specifies whether the agent can perform the action or not. 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.

Finally 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:

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

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

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:

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

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. In the table, we abbreviated *self* as s. Some discussion is needed about table 3.

request$_{out}$ **(j,** a **)** /* s denotes *self* */
pre = $\neg B_s G_j$ *Done*$(\langle j,a\rangle) \wedge B_s$ *Agent*$(j,a) \wedge I_s$ *Done*$(\langle j,a\rangle)$
post = revise($B_s B_j G_s$ *Done*$(\langle j,a\rangle)$). query(B_s *Trusts*(j,s)). revise($B_s G_j$ *Done*$(\langle j,a\rangle)$)
agree$_{out}$ **(j,** a **)**
pre = $B_s\alpha \wedge \neg B_s Bif_j \alpha$ **post** = revise($B_s B_j \alpha$), where $\alpha = G_s$ Done$(\langle s,a\rangle)$
refuse$_{out}$ **(j,** a **)**
pre = $B_s\neg$ *Feasible*$(a) \wedge B_s B_j$ *Agent*$(s,a) \wedge B_s\alpha \wedge \neg B_s Bif_j\alpha$
post = revise($B_s B_j \neg$ *Feasible*$(a) \wedge B_s B_j \alpha$), where $\alpha = \neg$ *Done*$(\langle s,a\rangle) \wedge \neg G_s$ *Done*$(\langle s,a\rangle)$
failure$_{out}$ **(j,** a **)**
pre = $B_s\alpha \wedge \neg B_s Bif_j\alpha$ **post** = revise($B_s B_j \alpha$)
α = *Done*$(\langle s,-\rangle$, *Feasible*$(a) \wedge G_s$ *Done*$(\langle s,a\rangle)) \wedge \neg$ *Done*$(\langle s,a\rangle) \wedge \neg G_s$ *Done*$(\langle s,a\rangle)$

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

In $request_{out}(j,a)$ the precondition is similar to those in the table 1 without $FP(a)[s\backslash j]$, and moreover *self* intends that a to be done. In its post-action, *self* will believe that j believes that *self* has a goal to do a and if *self* knows that j trusts *self*, it will imply that j will create a goal to do a. In contrast with table 1, here we do not consider $FP(a)[self\backslash j]$ for the sake of simplicity.

The preconditions of $agree_{out}$ are the same as those of $inform_{out}$ (defined before) but ω is replaced with α, which is defined as $\alpha = G_{self}$ *Done*$(\langle self,a\rangle)$ and its post-action simply adds a new formula $B_{self} B_j G_{self}$ *Done*$(\langle self,a\rangle)$. In the precondition of $failure_{out}$, *Done*$(-$,*Feasible*$(a) \wedge G_{self}$*Done*$(a))$ means that an action has been done and before performance of that action, a was feasible and *self* had a goal to do a, although in both of the performatives $refuse_{out}$ and $failure_{out}$, *self* does not intend any more to do a (it is shown in the definition of α).

4.1.2 Input action

Assume $perf_{in}(i,\alpha)$ is a P_act of agent j, where i is the sender, ρ=*post*($perf_{in}$(i, $\alpha\theta_j\eta$))*,* and η is a valuation of the free variables of $\alpha\theta_j$, which will be initialized during the communication. This will be clarified in the parallel composition rule. The semantics of receiving is defined as:

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

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

post(request$_{in}$(i, a)) /* s denotes *self* */
revise($B_s G_i$ $Done(\langle s, a\rangle)$).
{ query($B_s Feasible(a)$). revise($I_s Done(\langle s, a\rangle)$)
+ query($B_s \neg$ $Feasible(a)$). revise($\neg G_s$ $Done(\langle s, a\rangle)$) }
post(agree$_{in}$(i, a))
revise($B_s G_i$ $Done(\langle i, a\rangle)$)
post(refuse$_{in}$(i, a))
revise($B_s \alpha$). revise($\sim B_s B_i G_s$ $Done(\langle i, a\rangle)$)
where $\alpha = \neg$ $Feasible(\langle i, a\rangle) \wedge \neg$ $Done(\langle i, a\rangle) \wedge \neg G_i$ $Done(\langle i, a\rangle)$
post(failure$_{in}$(i, a))
revise($B_s B_i$ $\alpha \wedge B_s \alpha$). revise($\sim B_s B_i G_s$ $Done(\langle i, a\rangle)$)
α = $Done(\langle i, -\rangle$, $Feasible(a) \wedge G_i$ $Done(\langle i, a\rangle)$) $\wedge \neg$ $Done(\langle i, a\rangle) \wedge \neg G_i$ $Done(\langle i, a\rangle)$.

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

Let us explain the meaning of the P_acts of the table 4. After receiving *request(a)*, the receiver *self* will come to believe that i intends a to be done by *self* and it believes that i believes that *self* is able to do 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. In post-action of $agree_{in}(i,a)$, the receiver *self* will believe that i has intended to do a. In the $refuse_{in}(i,a)$, *self* believes that i believes α and *self* believes α too. Moreover *self* does not believe that i believes that *self* intends a to be done by i (see $request_{out}$ for understanding the reason for this). In this post-action α states that a is not feasible and it is not done by i and it is not intended any more to be done. The post-action of $failure_{in}$ is similar to $refuse_{in}$ but with a different definition of α.

4.2 Summation and Parallel composition

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

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

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

In the semantics of parallel composition there are two cases:

1: For any $l \in \{\tau\} \cup \ literals$:

$$\frac{\langle \Pi_i, \Psi_i, \theta_i, \alpha \rangle \rightarrow \langle \Pi_i', \Psi_i', \theta_i', did(i,l) \rangle}{[...|\langle \Pi_i, \Psi_i, \theta_i, \alpha \rangle|...] \rightarrow [...|\langle \Pi_i', \Psi_i', \theta_i', did(i,l) \rangle|...]}$$

2: Let $\bar{c}(perf(\alpha))$ be the message to be sent by i, and $c(perf(\beta))$ be the message to be received. 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:

$$\frac{\langle \Pi_i, \Psi_i, \theta_i, \alpha \rangle \rightarrow \langle \Pi_i', \Psi_i', \theta_i', did(i,\bar{\gamma}) \rangle \ , \ \langle \Pi_j, \Psi_j, \theta_j, \beta \rangle \rightarrow \langle \Pi_j', \Psi_j', \theta_j', did(j,\gamma) \rangle}{\begin{array}{c}[...|\langle \Pi_i, \Psi_i, \theta_i, \alpha \rangle|...|\langle \Pi_j, \Psi_j, \theta_j, \beta \rangle|...] \rightarrow \\ [...|\langle \Pi_i', \Psi_i', \theta_i', did(i,\bar{\gamma}) \rangle|...|\langle \Pi_j', \Psi_j', \theta_j', did(j,\gamma) \rangle|...]\end{array}}$$

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

5 Example: FIPA request Protocol

We now explain FIPA request protocol in the context of our framework. We will show the evolution of the IS during the running of the protocols. 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 presents the ECCS code of this protocol. In this 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 $B_i p$, and $Goal\ i\ p$ instead of $G_i p$.

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 state of agent i does not have formula 2 (because of $B_i\ Done(\langle j, a \rangle)$ and formulas 4 and 5 because of the post-action of *r_inform*. In the last state of j, formulas 3, 5 and 6 are deleted because of the addition of 8, semantics of post-action of *s_inform*, and addition of 7 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. We

```
Global function Agt = {(j,a)=true }
```
Agent i:
```
Initial_beliefs ={ Bel i Trustable(j) }
Initial_goals ={ Goal i Done(<j,a>) }
Comm_channels ={ c=(i,j), d=(j,i) }
Main program:
  query(Goal i Done(<k,act>)).    /* act is bound to a, and k is bound to j */
  (i,k)!(request(act)). /* instead of channel name we write its participants here */
  { (k,i)(agree(act)) +
    (k,i)(refuse(act))
  }.
  { query(Bel i (Goal j (Done<j,act>))).
    { (k,i)(inform(Done(act))) +
      (k,i)(failure(act)) }
  }
```

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

Fig. 1. FIPA request protocol, implemented in ECCS.

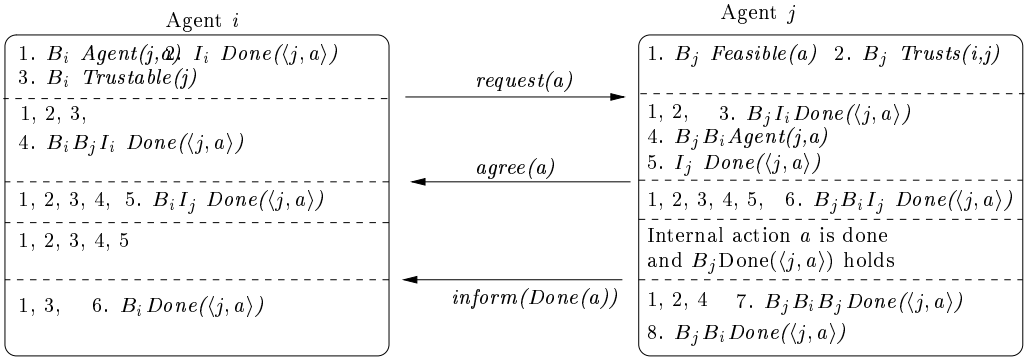

Fig. 2. FIPA-request protocol.

think this is one step to go in such a direction. The main contribution 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.