http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 7 (4): 631-638, 2008
ISSN 1812-5638

PRI-Based Communication and Collaboration Between Agents

Liguo Liu and Jianhua Zou
System Engineering Institute, Xi'an Jiaotong University, 28 Xianning West Road, Xi'an, 710049, China

Abstract: In this study, an extension to the original 3APL is made for the purpose of practical development of communication and collaboration between agents. Two pairs of primitives are incorporated into 3APL to implement information exchange and function collaboration between agents. Three basic actions dealing with priority are introduced to the agents to handle the tasks that require the priorities of collaborating candidates. The formal syntax and semantics of these primitives and basic actions as well as an example are presented.

Key words: Intelligent agent, agent-orient programming, communication between agents, collaboration between agents

INTRODUCTION

In most theories on agents, logic is the common facility to describe an agent and several concepts are incorporated into such logics such as beliefs, desires, intentions, commitments, goals and plans (Rao and Georgeff, 1991; Shoham, 1993; Hoek *et al*., 1998). It has been argued that it can be useful to analyze and specify a system in terms of these concepts (Rao and Georgeff, 1995). On the other hand, for the practical development of intelligent agents, several programming languages have been introduced and 3APL is one of these languages. A detailed description of 3APL is given in Hindriks *et al*. (1999) and Dastani *et al*. (2004). As an extension, communication between agents is integrated into 3APL in Hindriks *et al*. (2000). However, Hindriks *et al*. (2000) does not concern the priority of the collaborating agents so this approach is not very suitable for the situations that the priority is required. Else, the primitives defined in Hindriks *et al*. (2000) are somewhat ambiguous in our opinion, especially for the primitive ask. In this study, we are trying to extend the capabilities of an agent to deal with those situations mentioned above and give another set of more deterministic primitives as an improvement to the 3APL.

OVERVIEW OF 3APL

Here, we give an overview of the agent programming language 3APL for single agent described in Hindriks *et al*. (1999) and Dastani *et al*. (2004).

Agent: An intelligent agent in 3APL is an entity consisting of beliefs, goals and practical reasoning rules.

Beliefs are many formulae from some logical language L and represent the current environment and situation from the point of view of an agent. Language L is used for the purpose of knowledge representation so that it is the domain language.

Goals are kind of like imperative programs and supply the agent with all kinds of capabilities. The execution of goals updates the beliefs of an agent by adding and deleting information and further changes the viewpoints of the agent about its environment and states. Goals are defined on the basis of basic actions. Basic actions can be viewed as simple goals and specify the capabilities with which the agent can achieve a particular state of affairs and affect its beliefs. The execution of a goal is actually the execution of basic actions composing the goal.

The main purpose of practical reasoning rules is to supply the agent with a facility to manipulate its goals. Generally, a practical reasoning rule has the form of $\pi_h \leftarrow \Psi | \pi_b$ and consists of three components: a head π_h, a guard Ψ and a body π_b (Hindriks *et al*., 1999). If a goal that an agent pursues matches the head of a practical reasoning rule and the conditions in the guard are satisfied by the current beliefs, the agent may consider replacing the goal with the body of the rule, which implies the goal can be achieved by executing the body of the rule. A plan is a sequence of sub goals which may be executed directly or replaced by other plans. Therefore, the goals and the plans are equivalent to some extent in 3APL.

Now, some symbols used in this study and their meanings are shown in Table 1. A substitution θ is a ground substitution if for every pair $X = t \in \theta$ that term t is ground, i.e., $Free(t) = \emptyset$.

Corresponding Author: Liguo Liu, System Engineering Institute, Xi'an Jiaotong University, 28 Xianning West Road, Xi'an, 710049, China

Table 1: Symbols and their meanings

Symbols	Meanings for an agent
L	A domain language
Goal	A set of goals
Π	= {π_0, …, π_{i-1}, π_i, π_{i+1}, ….}; the current set of plans; a subset of Goal
σ, σ'	A set of beliefs; i.e., a belief base
Γ	A set of practical reasoning rules
<Π, σ, Γ>	An agent in 3APL.
<Π, σ>	Dynamic part of an agent; mental state.
θ, θ'	Substitution; a set of bindings of free variable X to object t in L.
Dom(θ)	A set of bound variables in θ
Free(Ψ)	A set of free variables in expression Ψ
<Π, σ, θ>	Configuration. be used to define the transition system
|=	Usual logical consequence operator.
E	Successful termination of a plan
BACT	A set of basic actions
T(a, b)	A transition function: BACT × L → L
Ψθ, φγ	Bindings of free variable(s) in Ψ and φ with the substitution θ and γ.

Transition and semantics: The running of an agent implies the changes of its mental states and the semantics specifies how an operation to influence on the mental states. In 3APL, it is an operational semantics and is defined by means of transition systems. A transition corresponds to a single computation step and a transition system consists of a set of transition rules for deriving the transitions. The general format of a transition rule consists of a set of premises which are transitions and a conclusion derivable from these premises which also is a transition. There are two levels of execution defined in 3APL, the agent level and the goal level. Corresponding to the two levels of execution, we now give two examples of the transition rule defined in 3APL, respectively. Further explanations can be referred to Hindriks *et al.* (1999).

Let V = Free(Π) is the set of global free variables occur in Π. Then

$$\frac{<\pi_i,\sigma,\theta>_V \rightarrow <\pi'_i,\sigma',\theta'>}{<\{\pi_0,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots\},\sigma,\theta>_V \rightarrow <\{\pi_0,\ldots,\pi_{i-1},\pi'_i,\pi_{i+1},\ldots\},\sigma',\theta'>} \quad (1)$$

This transition rule indicates that the semantics of an agent is defined on the basis of the semantics of a single goal π_i, which is selected to execute and is updated to π'_i after the computation step. As a discriminator, the set variable V is used to prevent implicit binding for the new-added variables.

For the goal execution, an example of these transition rules is given here, which is for the execution of basic actions.

Let α(t) ∈ BACT be a basic action.

$$\frac{T(\alpha(t)\theta,\sigma)=\sigma'}{<\alpha(t),\sigma,\theta>_V \rightarrow <E,\sigma',\theta>} \quad (2)$$

where, α(t)θ is to bind the free variables in α(t) with the substitution θ. The execution of α(t) modified the beliefs of the agent, which is shown by the changed beliefs, from σ to σ'.

COMMUNICATION AND COLLABORATION

A multiple agents system is a definite set of agents $\{A_1,..,A_i,\ldots,A_j,\ldots,A_n\}$ and at the multi-agent level, the interactions between agents can be defined as two types roughly: the communication of beliefs and the communication of goals. By and large, the former can be viewed as information exchange and the latter as function exchange. For each type of communication, we propose a pair of primitives correspondingly. The pair of inform and learn is for the former and the pair of request and respond is for the latter. The functions existing in each primitive should be viewed as an atomic function and be executed uninterruptedly. The semantics of these primitives is also an operational semantics defined by transition rules (El Fallah-Seghrouchni and Suna, 2004) and is based on two types of reasoning: deduction and abduction. Deduction serves to derive information from a received message and abduction (Poole, 1989) serves to explain facts or to build plans for achieving goals. The reasoning involved in inform and learn is deduction while those involved in request and respond are both of them.

The primitives defined here are synchronous communication primitives which demand the simultaneous participation of both agents. Note that by using buffers, asynchronous communication can be simulated by synchronous communication.

Prior to the detailed explanation of the primitives, we firstly define the semantics of a computation step for the interaction between two agents in a multi-agent system.

$$\frac{A_i \leftrightarrow A_j, A_i \rightarrow A'_i, A_j \rightarrow A'_j, 1 \le i \ne j \le n}{\{A_1,\ldots,A_i,\ldots,A_j,\ldots,A_n\} \rightarrow \{A_1,\ldots,A'_i,\ldots,A'_j,\ldots,A_n\}} \quad (3)$$

where, $A_i \leftrightarrow A_j$ implies A_i and A_j interact with each other, $A_i \rightarrow A'_i$ implies agent A_i transforms into A'_i after the interaction and A_j transforms into A'_j. After the computation step, the MAS turns into $\{A_1,..,A'_i,\ldots,A'_j,\ldots,A_n\}$.

Information exchange: Information exchange implies the communication of beliefs between agents and the primitives for this purpose are inform and learn. Firstly we give their qualitative definitions.

Inform is a primitive for information exchange and is used by the sending agent. It serves the purposes: packing a message and sending the message.

Learn is a primitive for information exchange and is used by the receiving agent. It serves the purposes: receiving a message and deducing the useful information from the message.

The reasoning that learn employs is deduction, which serves to derive information from a received message.

For the information exchange, the sending agent is called a consultee and the receiving agent a learner. We differentiate two cases as follows. On the one hand, if learner has asked consultee a question, it should adopt the primitive learn as its next action to receive the message sent by consultee. On the other hand, if learner only plays as a receiver to receive the message sent by consultee, such as the scenario of subscribing-delivering, for the purpose of deriving some useful information from the message received, learner should also adopt the primitive learn as its action to receive this message as if it has a virtual question to address. Hence, the information can be accepted if the answer to the (virtual) question is deduced from the received message; otherwise, it will be discarded.

In the sequel, the semantics of these two primitives is introduced by means of transition rules. Note that for the clarity, the transition rules below for inform and learn are unilateral computational steps and just for the consultee and the learner, respectively. Since the communication is synchronous between agents, actual communication, which is called a handshake, only occurs when the consultee sends the information and the learner receives and accepts the information.

The expression inform(α, Ψ) specifies an communicative operation, which packs a message and attempts to send the message Ψ to the learner α. Because the purpose of inform(α, Ψ) is to share information with α, it requires that the expression Ψ must have been grounded by the substitution θ of the consultee before it is transmitted, that is, Free (Ψ) is empty. The execution can not affect the current states of the consultee, which is shown by the unchanged belief base σ and substitution θ.

$$\frac{\sigma \models \psi\theta}{<\text{inform}(\alpha,\psi),\sigma,\theta> \rightarrow <E,\sigma,\theta>} \quad (4)$$

The meaning of each symbol is referred to Table 1, where, $\sigma \models \Psi\theta$ means that Ψ is bound by the substitution θ and is satisfied by the beliefs σ of the consultee. Similar explanation is suitable for the following transition rules.

The expression learn (β, φ) defines another communicative operation, receiving a message from consultee β and deriving an answer to the (virtual) question φ. In this case, free variables in the question φ indicate what the question is about. A question φ addressed means that the expression $\sigma \cup \Psi \models \varphi$ is satisfied, of which σ is the belief base of the learner and Ψ is the message supplied by the consultee. The answer to the question φ is the set of bindings γ to the free variables occurring in φ, where γ is a new substitution and can be used in the further computation by the learner. Consequently, the execution of this action will affect the belief base and the substitution of the learner as follows.

$$\frac{\sigma \bigcup \psi \models \varphi\gamma}{<\text{learn}(\beta,\varphi),\sigma,\theta> \rightarrow <E,\sigma',\theta \bigcup \gamma>} \quad (5)$$

As mentioned earlier, an actual information communication (a handshake) occurs when the consultee sends the information and the learner receives and accepts it. We can specify this situation as the following rule.

Let A be the consultee, B be the learner, Ψ be the message sent by A, φ be the (virtual) question and σ be the belief base of B.

$$\frac{A \leftrightarrow B, A \rightarrow A', B \rightarrow B', \sigma \bigcup \psi \models \varphi}{\{A,B\} \rightarrow \{A',B'\}} \quad (6)$$

where, the symbols such as $\leftrightarrow$ have the same meanings in Eq. 3.

Function collaboration: This case occurs when an agent needs functional assistance from other agent. The pair of primitives, request and respond, are introduced for this purpose. As done for inform and learn, their qualitative definitions are given firstly.

Request is the primitive for function exchange and is employed by the requesting agent. It serves three purposes: requesting an explanation for an observation; requesting an answer to a question and requesting the implementation for an unrealized goal.

Respond is the primitive for function exchange and is employed by the responding agent. Corresponding to request, it also serves three purposes: abducing an explanation to an observation; deducing an answer to the requested question and abducing a plan to implement a goal.

Here, we call a requesting agent an applicant and a responding agent an assistor. An observation can be viewed as an occurred goal, i.e., a fact.

Request allows the agent to apply for some functions and respond allows the agent to make an offer in response to a request. The primitive respond does not inform the applicant of the offer even if the assistor acquires it. Informing the applicant is the duty of inform. The reasoning involved in primitive respond is both abduction, by which the assistor abduces a cause to an effect or a plan for a goal and deduction, by which the assistor deduces an answer to the question raised by the applicant. Abduction (Poole, 1989) is informally paraphrased as reasoning from an observation to an explanation, or alternatively, from an effect to a cause. Abduction reasoning not only can be used to give a cause of an observation (Shanahan, 1989), but can be used to build a plan for the achievement of a goal.

The semantics of these two primitives is also introduced by means of transition rules. The expression request (α, Ψ) specifies a request to assistor α to explain, answer or achieve Ψ, where Ψ may be an observation, a question or an unrealized goal. In this study, we stipulate that all three kinds of requests are function collaborations between agents and specially, the request that building a plan to achieve a goal is an applying for a service. If it can fulfill the goal based on its current configuration, the assistor should collaborate with the applicant. As explained for inform and learn, the transition rules for request and respond below are unilateral computational steps. A handshake between agents only occurs if the applicant sends the request and the assistor abduces or deduces a specific offer that satisfies the request.

After execution of request (α, Ψ), the offer that the assistor made is not automatically informed to the applicant, that is, it receives no information. It is illustrated by the unchanged belief base σ and substitution θ of the applicant.

Let Ψ be the goal or the observation that the applicant requests to complete or to explain and α be the assistor.

$$\frac{\sigma \models \psi\theta}{< \text{request}(\alpha,\psi),\sigma,\theta > \rightarrow < E,\sigma,\theta >} \quad (7)$$

Let Ψ be the question that the applicant expects to get an answer and α be the assistor.

$$\frac{\sigma \not\models \psi}{< \text{request}(\alpha,\psi),\sigma,\theta > \rightarrow < E,\sigma,\theta >} \quad (8)$$

where, $\sigma \not\models \Psi$ means that Ψ is something that the applicant does not know and wants to get an answer.

The execution of the expression respond (β, φ) means to make an offer φ in response to an applicant β. Corresponding to each of the three purposes of request, a transition rule is specified for respond, respectively. Note that below γ corresponds to a substitution restricted to the free variables of φ, that is,

$$\text{Dom}(\gamma)=\text{Free}(\varphi) \text{ and } \text{Dom}(\gamma) \cap \text{Dom}(\theta)=\varnothing$$

For the case that request is employed to request an explanation of an observation, the assistor needs to abduce a proper grounded offer φ based on which the observation Ψ is caused. That is, the assistor computes the substitution γ such that $\sigma \cup \varphi\gamma \models \Psi$.

For the case that the applicant requests a service from the assistor, Ψ is grounded and represents a goal for which the applicant is incompetent. The request is satisfied only if the plan for this goal is built and executed successfully. So, the execution of respond(β, φ) can be viewed as to build the plan and place it in the plan base of the assistor. In this case, φ denotes the offer that the assistor whether or not can fulfill the goal and the detailed plan is denoted by ξ in the transition rule.

For the case that request is employed to ask the assistor a question Ψ. Let $\varphi = \Psi$ now. The duty of the assistor is to deduce an appropriate answer to the question φ on the basis of its own beliefs. That is, what the assistor needs to do is computing the substitution γ that satisfies the expression $\sigma \models \varphi\gamma$.

We now specify the first form of the transition rule for respond (β, φ). In this case, the assistor obtains a new substitution γ which can be utilized in further computation. The modification of the substitution, $\theta \rightarrow \theta \cup \gamma$, proves it as follows.

Let Ψ be the observation that the applicant requests to explain.

$$\frac{\sigma \models \varphi\gamma, \sigma \cup \varphi\gamma \models \psi}{< \text{respond}(\beta,\varphi),\sigma,\theta > \rightarrow < E,\sigma,\theta \cup \gamma >} \quad (9)$$

where, the premise $\sigma \cup \varphi\gamma \models \Psi$ means that the offer φ is grounded and is a possible cause of Ψ.

For the case that the applicant requests a service from the assistor, the assistor manages to build a plan ξ for the goal Ψ and a new substitution γ can be acquired and utilized in further computation. These are shown in the following rule.

Let Ψ be the goal that the applicant requests the assistor to complete.

$$\frac{\sigma \models \varphi\gamma, \sigma \cup \varphi\gamma \models \psi}{< \text{respond}(\beta,\varphi),\sigma,\theta > \rightarrow < \xi,\sigma,\theta \cup \gamma >} \quad (10)$$

where, the plan ξ is placed into the goal base of the assistor to be scheduled as soon as possible.

Next we define the third form of the transition rule for respond (β, φ). For answering the question φ (= Ψ), the variables in it should be bound and thus, a substitution γ may be obtained by the assistor. These variable bindings, however, are only available to the variables in the question φ and inapplicable to the variables occurs in the assistor. Consequently, the substitution γ should not be introduced into the assistor.

Let φ (= Ψ) be the question raised by the applicant.

$$\frac{\sigma \models \varphi\gamma}{<\text{respond}(\beta,\varphi),\sigma,\theta>\rightarrow<E,\sigma,\theta>} \quad (11)$$

As mentioned earlier, a handshake between agents only occurs when the applicant sends the request and the assistor is able to satisfy it. For all kinds of the transition rules of primitives request and respond, their handshake can be described as the transition rule below.

Let A be the applicant, B be the assistor, Ψ be the request sent by the applicant, φ be the offer made by the assistor and σ be the belief base of the assistor.

$$\frac{A \leftrightarrow B, A \rightarrow A', B \rightarrow B', \sigma \bigcup \varphi \models \psi}{\{A,B\} \rightarrow \{A',B'\}} \quad (12)$$

where, the sub-expressions have the same meanings as those in Eq. 6.

PRIORITY

Just as mentioned earlier, we now introduce priorities into agents for the situations requiring the order of the participants in cooperation. The processing of priority is in essence an internal operation for an agent since its running does not require the participation of other agent generally. On the one hand, it can be regarded as a preprocessing when an agent selects an appropriate collaborating fellow in its collaborating circle based on the priority and on the other hand, it can be viewed as a postprocessing if an agent needs to reassign a new priority to its collaborating fellow in line with the collaboration. Here we assume that an agent has held the priorities for its collaborating candidates and can use them directly. The detailed computation of priority is beyond the scope of this study and will be discussed in another topic.

Correspondingly, the selecting of the most suitable collaborating fellow is defined as a basic action, sort(Ag, Cir) and the evaluating of collaborating fellow is defined as two: eval(Ag, Res), modi(Ag, Res, Cir). Firstly we stipulate that a string started with an uppercase denotes a variable and, a string started with a lowercase denotes a constant. Specially, Cir is a set variable that records all available collaborating candidates, which is called the collaborating circle of an agent. We specify it as a part of the beliefs of an agent. Below we give the semantics of these three basic actions by means of transition rules.

$$\frac{T(\text{sort}(Ag, Cir),\ \sigma) = Ag(hw),\ \sigma \models Cir\theta,\ \ Cir \models hw}{<\text{sort}(Ag, Cir),\ \sigma,\ \theta> \ \rightarrow \ <E,\ \sigma \bigcup \{Ag(hw)\},\ \theta>} \quad (13)$$

$$\frac{T(\text{sort}(Ag, Cir),\ \sigma) \ = \ Ag(null),\ \ \sigma \models Cir\theta}{<\text{sort}(Ag, Cir),\ \sigma,\ \theta> \ \rightarrow \ <E,\ \sigma \bigcup \{Ag(null)\},\ \theta>} \quad (14)$$

where, hw is the candidate with the highest priority in Cir. The expressions Ag(hw) and Ag(null) assign hw and null to Ag, respectively and thus, Ag represents the selected collaborating fellow. σ |= Cirθ implies that Cir exists in σ while Cir |= hw means that hw is an elements of Cir. The second rule is for the case either the Cir is empty or there is no suitable candidate in the Cir.

$$\frac{T(\text{eval}(Ag,Res),\ \sigma) = Res(Ag),\quad \sigma \models Ag\theta}{<\text{eval}(Ag, Res),\sigma,\ \theta> \ \rightarrow \ <E,\ \sigma \bigcup \{Res(Ag)\},\ \theta>} \quad (15)$$

where, the expression σ |= Agθ specifies that Ag denotes the collaborating fellow while the one Res(Ag) means that Res is the new priority of Ag.

$$\frac{\begin{array}{l}T(\text{modi}(Ag, Res, Cir),\ \sigma) \ = \ \{Cir|<Ag\ ,\ Res>\},\\ \sigma \models Cir\theta,\ Res\theta,\ Ag\theta\end{array}}{<\text{modi}(Ag, Res,Cir),\ \sigma,\theta) \ \rightarrow \ <E,\ \sigma \bigcup \{Cir|<Ag,\ Res>\},\theta>} \quad (16)$$

where, the expression σ |= Cirθ, Resθ, Agθ implies that Cir, Res and Ag are all exist in σ and the one {Cir|<Ag, Res>} implies that replacing the priority of Ag in Cir with the new priority, Res.

By means of these rules, one agent is endowed with the capabilities to select the collaborating fellow based on the priorities and to evaluate the performance of its collaborating fellow after the collaboration.

EXAMPLE

Goals for service collaboration: Based on the primitives defined above and their respective transition semantics, combining with the processing of priorities, we can now define some complex goals for the service requesting and supplying to demonstrate their usage. The stipulation

about the definition of variables and constants in last section is still available. For the applicant, the sorting and selecting of the collaborating candidates should be done prior to the actual collaboration while the evaluating of the collaborating fellow should be completed after the collaboration. Correspondingly, two goals are introduced and they are SORT_AGENT(Ag, Cir), which is defined to select one suitable collaborating fellow Ag from the collaborating circle Cir and ASSESS(Ag, Cir), which is employed to reassign the priority of Ag and adjust the collaborating circle Cir for the future collaboration. Also, other interactions between agents must be introduced, for example, querying whether or not an agent can supply a function, negotiating the specific parameter assignments for a function, launching the function and so on. Here, the three kinds of processes mentioned are completed by the goals REQUEST_SERV and REPLY_SERV, NEGOTIATE_PARA and CONTINUE_NEGO, CONDUCT, respectively. Each of these goals can be viewed as an abstract processing procedure in the imperative programming. As complex goals, they are all implemented by those three basic actions, those four primitives and themselves. Below, the practical reasoning rules for these goals are specified, in which the body of a rule is the plan for the achievement of the goal. Based on those primitives and basic actions, these rules are pellucid.

```
SORT_AGENT(Ag, Cir)
 ← true | sort(Ag, Cir);
        IF Ag is empty
           null;
        ELSE
           REQUEST_SERV(Ag, Func);
```

where, null is a placeholder standing for the drop of the collaboration.

```
REQUEST_SERV(Ag, Func)
 ← true | request(Ag, Func);
        learn(Ag, Allow_or_not);
        IF Allow_or_not is allowed
           NEGOTIATE_PARA(Ag, Set(Para_list));
        ELSE
           SORT_AGENT(Ag', Cir);
        END
REPLY_SERV(Ag, Request_result)
 ← true | respond (Ag, Request_result);
        inform(Ag, Request_result);
```

where, Func is the goal (function) that the applicant applies. Request_result is used to show whether or not the assistor is able to collaborate with the applicant. Ag' means to find another collaborating fellow by invoking SORT_AGENT.

```
NEGOTIATE_PARA(Ag, Set(Para_list))
 ← true | request(Ag, Set(Para_list));
        learn(Ag, Nego_result);
        IF Nego_result is accepted
           CONDUCT(Ag);
        ELSE IF Nego_result is not_accepted
           CONTINUE_NEGO (Ag, Con);
        ELSE
           SORT_AGENT(Ag, Cir);
        END
CONTINUE_NEGO(Ag, Con)
 ← true | respond (Ag, Con);
        IF Con is accepted
           inform(Ag, accepted);
           CONDUCT(Ag);
        ELSE IF Con is not_accepted
           inform(Ag, not_accepted);
        NEGOTIATE_PARA(Ag, Set(Para_list'));
     ELSE IF Con is cancel
        SORT_AGENT(Ag, Cir);
     END
```

where, Para_list' consists of the parameter assignments that need further negotiation. Con is the decision about the parameter negotiation made by an agent and, Set(Para_list) is the auxiliary procedure to configure the parameters needed by Func, in which Para_list is the list that consists of the parameter assignments.

```
CONDUCT(Ag)
 ← true | inform(Ag, start);
        learn(Ag, Act);
        IF Act is start
           Launch()
        ELSE
           CONDUCT(Ag)
        END
```

where, Launch() means to start the actual function collaboration simultaneously by the collaborating agents. It is only a placeholder here and in the practical programming, can be replaced by the goal that fulfills the function indeed. Else, start is the prompt to notify the opposite agent of the startup of Launch().

```
ASSESS(Ag, Cir)
 ← true | eval(Ag, Res);
        modi(Ag, Res, Cir)
```

Step0 (agent ca):
Π: {SORT_AGENT (Ag, Cir);|}
σ(Cir): {<cd, 0.8>; <cd, 0.65>} θ: {Fune = fune_name;}
Γ: {SORT_AGENT (Ag. Cir)
← true | sort (Ag, Cir);
IF Ag is empty
null;
ELSE
REQUEST_SERV (Ag, Func);}
Initially,
SORT_AGENT (Ag, Cir): the only available plan;
<cb, 0.8>: a collaborating candidate, cb and its priority, 0.8;
<cd, 0.65>: a collaborating candidate, cd and its priority, 0.65;
Func = func_name: the applying setvice name, func_name, is bound to the variable Func;
Γ: the five rules defined above are dept. Here, only one is listed and others are omitted;
Basic actions are not listed here.

Agent ca schedules the only plan (goal) SORT_AGENT (Ag, Cir) and matches it with the practical reasoning rules. The practical reasoning rule SORT_AGENT (Ag, Cir) is matched and the guard of the rule is satisfied (it is always satisfied because the guard is true). Thus, as a transition reasoning result, the plan SORT_AGENT (Ag, Cir) is replaced by the body of the practical reasoning rule and the states of agent ca are listed in the next step.

Step 1 (agent ca):
Π: {sort(Ag, Cir);
IF Ag is empty
null;
ELSE
REQUEST_SERV (Ag, Func);}
σ/θ/Γ: unchanged
Agent ca tries to execute the new plan, i.e., the body of the practical reasoning rule. It schedules the basic action sort (Ag, Cir) and obtains a new binding Ag = cb as follows.

Step 2 (agent ca):
Π: {IF Ag is empty
null;
ELSE
REQUEST_SERV (Ag. Func);}
σ/θ/Γ: unchanged; θ: {Func_name; Ag = cb;}
Next, agent ca completes the condition sentence and appends the plan, REQUEST_SERV (Ag, Func), to the plan base Π. The variables in the new plan are bound by the substitution of agent ca.

Step 3 (agent ca):
Π: {REQUEST_SERV (cb, func_name);}
σ/θ/Γ: unchanged;

Step 4: (agent ca): ←→ Step 0 (agent cb)
Π: {request(cb, func_name);
learn(cb, Allow_or_not);
IF Allow_or_not is allowed
NEGOTIATE_PARA(cb, Set(Para_list));
ELSE
SORT_AGENT(Ag, Cir);
END
}
σ/θ/Γ: unchanged;
The states of agent ca after the expansion of plan REQUEST_SERV(cb, func_name).

Step 5 (agent ca):
Π: { learn(cb, Allow_or_not);
IF Allow_or_not is allowed
NEGOTIATE_PARA(cb, Set(Para_list));
ELSE
SORT_AGENT(Ag, Cir);
END
}
σ/θ/Γ: unchanged;
Agent ca schedules the plan request(cb, func_name) to request the collaboration with cb.

Step 6 (agent ca):
Π: {If Allow_or_not is allowed
NEGOTIATE_PARA(cb, Set(Para_list));
ELSE
SORT_AGENT(Ag, Cir);
END
}
σ: {< cb, 0.8 >; <cb, 0.65 >; request_res(cb, allowed)}
θ: {Func = func_name; Ag = cb; Allow_or_not = allowed;}
Γ: unchanged;
Agent ca learns that cb has approved the collaboration through the execution of learn(cb, Allow_or_not). The new contents in belief base and substitution reveal the requesting result.

Step7 (agent ca):
Π : { NEGOTIATE_PARA(cb, Set(Para_list); }
σ : { <cb, 0.8> ; <cd, 0.65>; request_res(cb, allowed)}
0 : { Func = func_name; Ag=cb; allow_or_not = allowed ; }
Γ : unchanged;
Append the new plan NEGOTIATE_PARA(cb, Set(Para_list)) to the plan base Π, Prepare for the parameter negotiation.

Step 0 (agent cb): ←→ Step4 (agent ca)
Π; {learn(Ag, Serv);}
σ/θ/Γ: unchanged
As an information receiver and a daemon, learu(Ag, Serv) is use to receive the service request from other agent.

Step 1 (agent cb):
Π; {REPLY_SERV(ca, Request_result)}
σ: {<service(ca, func_name)> }
θ: {serv = func_name; Ag = ca; }
Γ: unchanged
Receive the request from agent ca and modify the state accordingly, <service(ca, func_name)>, means that agent ca applies the service. The substitution records the new bariable bindings. REPLY_SERV(ca, Request_result) is the suitable goal to answer the applicant ca.

Step 2 (agent cb):
Π: { respond (ca, Request_result); }
inform(ca, Request_result); }
σ: {<service(ca, func_name; Ag = Cb; }
θ: {Serv = func_name; Ag = cb;}
Γ: unchanged
The new states of agent cb after the expansion of the goal REPLY_SERV(ca, Request_result).

Step 3 (agent cb):
Π: {}
σ: {<service(ca, func_name)>; sevice_offer(ca, allowed); }
θ: {Serv = func_name; Ag = cb; Request_result = allowed }
Γ: unchanged
After the execution of respond (ca, Request_result) and inform(ca, Request_resul), cb has accepted the collaboration request. Request_result = allowed and service_offer(ca, allowed) reveal its decision.

Fig. 1: Processing for a service applying

Service applying: Given these goals for service collaboration above, we now describe a practical processing for a simple example as an exhibition of their utilization. For the brief, we only involve SORT_AGENT(Ag, Cir), REQUEST_SERV(cb, Func) and REPLY_SERV(ca, Request_result), i.e., completing a service applying. Say agent ca has to apply a functional assistance from other agent. Moreover, we assume that there are more than one agents existing in the multi-agent system can serve agent ca, of which agent cb is one. Of course, agent ca is the initiator of the interaction. The processing is detailed in Fig. 1, where some appropriate explanations for the processing steps are given. The true interactions between ca and cb occur from step 4 for ca and step 0 for cb, which is the meaning of the symbols appended to these two steps.

DISCUSSIONS AND FUTURE RESEARCHES

The idea in this study is inspired by Hindriks *et al.* (2000). However, as an improvement, we give more distinct definitions and utilities of the primitives for communication and collaboration between agents, which are more rational and natural in our opinion. Moreover, we introduce priority into agent and employ it for the choice of collaborating candidate. As a utilization of the primitives and basic actions, we define several complex goals and give a simple example of service applying between agents. More complex and practical situations can be implemented by the similar processing. Another point to which we should pay attention is that we allow the applicant to select a collaborating fellow in this study and the same approach can also be used by an assistor to chooses an applicant when several applicants reach simultaneously. Future jobs include integrating both the capabilities and the priority of an agent into the collaborating mechanism and, the implementation of automatic beliefs diffusion and goals sharing among agents in order to realize the autonomy of MAS to some extent.

REFERENCES

Dastani, M., M.B. van Riemsdijk, F. Dignum and J.J. Ch. Meyer, 2004. A Programming Language for Cognitive Agents: Goal Directed 3APL. Programming Multi-Agent Systems. Springer Berlin, pp: 111-130.

El Fallah-Seghrouchni, A. and A. Suna, 2004. Programming mobile intelligent agents: An operational semantics. Proceedings of the Intelligent Agent Technology, IEEE/WIC/ACM International Conference on (IAT'04). Washington, DC, USA, pp: 65-71.

Hindriks, K., F.S. de Boer, W.V.D. Hoek and J.J. Ch. Meyer, 1999. Agent programming in 3APL. Auton. Agent Multi-AG., 2 (4): 357-401.

Hindriks, K., F.S. de Boer, M.B. van Riemsdijk and J.J. Ch. Meyer, 2000. Semantics of Communicating Agents Based on Deduction and Abduction. Issues in Agent Communication. Springer London, pp: 63-79.

Hoek, W.V.D., B. van Linder and J.J.Ch. Meyer, 1998. An integrated modal approach to rational agents. Foundations of Rational Agency, Applied Logic Series 14. Kluwer, Dordrecht, pp: 133-168.

Poole, D., 1989. Explanation and prediction: An architecture for default and abductive reasoning. Comput. Intel., 5 (2): 97-110.

Rao, A.S. and M.P. Georgeff, 1991. Modeling rational agents within a BDI-architecture. Proceedings of Knowledge Representation and Reasoning (KR and R-91). Morgan Kaufmann, pp: 473-484.

Rao, A.S. and M.P. Georgeff, 1995. BDI-agents: From theory to practice. Proceedings of the 1st International Conference on Multiagent Systems. San Francisco, pp: 312-319.

Shanahan, M., 1989. Prediction is deduction but explanation is abduction. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '89). San Mateo, CA, pp: 1055-1060.

Shoham, Y., 1993. Agent-oriented programming. Artif. Intel., 60 (1): 51-92.