# Requirement/Service Cooperation Model of Multi-Agent System in the Situation Calculus

**Liu Yisong [†,††], Zhong Shan [†] and Sun Yamin [††],**

[†]School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, 212013, China
[††]School of Computer Science & Technology, Nanjing University of Science & Technology, Nanjing, 210094, China

**Summary**
In multi-agent systems, the Requirement/Service is a cooperation way that is simple, efficient, and wildly applied. Under the framework of Situation Calculus, the agents' mental states (knowledge, task/goal, etc.) are represented by some special fluents and complex actions, and communication actions are introduced into the earlier ConGolog referring to FIPA-ACL. Consequently, we propose a Requirement/Service cooperation model and semantics for multi-agent System based on the extended ConGolog. What's more, in terms of the model and semantics, we specify a feasible case of the multi-agent system in the dynamic and incompletely known environment for achieving the cooperation based on reasoning about action and change.

***Key words:***
*situation calculus, multi-agent system, cooperation model, cooperation semantics*

## 1. Introduction

In multi-agent systems, the information and resources of system is dynamically changeable and incompletely known to every agent, and each agent's ability is limited. In order to achieve the given goal or task, the agents in system needs to cooperate efficiently [1,2]. There are various collaboration ways. In particular, the Requirement/Service is wildly applied in multi-agent systems [3].

McCarthy put forward the term "Situation Calculus" firstly in 1963. Ray Reiter formalized it, and brought out the Basic Theories of Action [4], and implemented an agent-oriented high level programming language (Golog[5], ConGolog[6], etc.), which made it practicable that reason about action and goal-oriented plan in dynamic environment.

ConGolog is extended from Golog that is appropriate for single agent, so agent's actions can be concurrently performed. However, ConGolog lacks mandatory communication predicates (actions), in particular, it cannot explicitly express the internal mental states of agents. Many theorists have done some significant works [7,8]. But they didn't put forward a complete and feasible multi-agent cooperation model and corresponding semantics.

In the Situation Calculus, we represent agents' mental states (knowledge, task/goal, etc.) by some special fluents (*Intend, Trigger*, etc.) and complex actions. In the earlier ConGolog, we add communication actions (*Inform, Request, Promise, Refuse and Result*) and point out how communication actions influence agent's mental states. What's more, we propose a novel Requirement/Service Cooperation Model and corresponding Semantics according to a method called Whole-hearted Satisfaction.

## 2. The Situation Calculus and ConGolog

As for the Situation Calculus, all changes to the world are the result of named actions. A possible world history, which is simply a sequence of actions, is represented by a first-order term called a situation. The constant $S_0$ is used to denote the initial situation, namely that situation in which no actions have yet occurred. There is a binary function symbol *do* and the term *do*(*a*, *s*) denotes the situation resulting from action *a* being performed in situation *s*. Action is denoted by function symbols, e.g. *pickup*(*agt,Coff*). Fluent denotes the value of world state in a certain situation, whose values vary from situation to situation, including relational fluents (e.g., *Holding*(*agt,Coff,s*)) and functional fluents (e.g., *Location*(*agt,s*)).

There are three domain dependent axioms. Action Precondition Axioms denotes conditions of action *a* being performed in situation s, e.g., *Poss(pickup(agt,*Coff*), s)* ≡ ¬*Holding(agt,*Coff*,s)* ∧ *Location(agt,s)=loc(*CM*)*. Successor State Axioms (including Effect Axioms and Frame Axioms), which specify how the action affects the state of the world (i.e., the value of fluent), for example, *Location(agt,do(A,s))=Loc* ≡ *A = goto(agt,Loc)* ∨ *(Location(agt,s)=Loc* ∧ ¬*A=goto(agt,Loc1))*. Axioms Describing the Initial Situation that is a set of first-order sentences that are uniform in $S_0$. Herein, the above three domain-dependent axioms, together with some domain

Manuscript received September 22, 2007
Manuscript revised September 25, 2007

independent axioms (e.g., Unique Names Axiom) constitute the Basic Theories of Action.

ConGolog has only two primitive actions: primitive actions $a$, test actions $\phi?$. Sequentially, suppose $\delta$, $\delta 1$ and $\delta 2$ stand for complex actions, complex actions of ConGolog can be defined recursively as: sequence $\delta 1; \delta 2$, nondeterministic choice of two actions $\delta 1 | \delta 2$, nondeterministic choice of action argument $(\pi x)\delta(x)$, nondeterministic iteration $\delta^*$, synchronized condition ***if*** $\phi$ ***then*** $\delta 1$ ***else*** $\delta 2$, synchronized loop ***While*** $\phi$ ***do*** $\delta$ ***EndWhile***, concurrency $\delta 1 \| \delta 2$, concurrency with $\delta 1$ at a higher priority $\delta 1 >> \delta 2$, concurrent iteration $\delta^{\|}$, interrupt $< \phi \rightarrow \delta >$. What's more, Procedures are defined with the syntax: ***Proc*** *P(v)* $\delta$ ***EndProc***.

The semantics of the ConGolog is a style of structural operational semantics. Based on two special predicates: Final and Trans, the overall semantics of program $\delta$ (i.e. complex actions or Procedures ) are defined using the *Do* abbreviation,

$$Do(\delta, s, s') \Box \exists \delta'(Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')) \quad (1)$$

Trans* is the reflexive transitive closure of the transition relation Trans. *Do( $\delta$ , s, s')* holds if it is possible to repeatedly single-step $\delta$ obtaining $\delta'$ and *s'* such that $\delta'$ can legally terminate in s'. In terms of this semantics, the ConGolog interpreter can automatically convert $\delta$ into a sequence of primitive actions using the mechanism called "regression". That is, if the agents' tasks, goals or intention are specifically described by $\delta$, they will be implemented when the agent reaches situation *s'* from situation *s* by executing the corresponding sequence of primitive actions.

## 3. Agents' Mental States and Communication Actions

Communication is the foundation of the cooperation. Communication actions does not generally have an effect on the environment that agent is within, but on agent's mental states, while the mental states determine the agent's behaviors. We specify some special fluents such as *Intend, Trigger* to represent agent's mental states, and introduce communication actions to ConGolog referring to the Agent Communication Language FIPA-ACL. The main communication actions include *Inform*, *Request*, *Promise*, *Refuse* and *Result*.

*Inform(i,j,inf( $\phi$ ))* denotes agent *i* inform agent *j* of a message $\phi$. The function *Inf( $\phi$ )* denotes $\phi$ is a message.

*Request(i,j,Task(c, $\delta$ ))* denotes agent *i* request agent *j* to achieve the task $\delta$ for agent *c*. The function *Task(c, $\delta$ )* denotes $\delta$ is a task, and expressed by complex actions or Procedures. The influence of action *Request* on the mental states of requestee agent *j* can be thought as whether the request is transformed to a task/goal of the requestee agent *j*. The transformation process may be rather complicate. Many aspects have to be considered, for example, whether the requestee agent *j* understands (has desire to achieve, and has ability to achieve) the request $\delta$, and whether the request is in conflict with the current task of agent *j*.

A feasible method is to introduce a special fluent *Intend(j,Task(c, $\delta$ ),s)* to denote that agent *j* intend to perform the task $\delta$, and suppose that a agent can carry out only one task at a time. If the requestee agent *j* is "free", action *Request* will make fluent *Intend* hold, whereas is the otherwise case. Besides, action *Request* can attach action *Inform* to send a message, for example, "Please give me a cup of coffee".

*Promises(i,j,Task(c, $\delta$ ))* denotes agent *i* promises to achieve the task $\delta$ from agent *j*. The precondition action *Promises* being performable is that *Intend(i,Task(k, $\delta$ ),s)* is hold. Action *Promises* will trigger agent *i* to perform $\delta$. Introducing a special fluent *Trigger(i,Task(c, $\delta$ ),s)* is to denote whether agent *i* be triggered to perform the task $\delta$. Action *Promises* will make fluent *Trigger* hold, and the interrupting action *<Trigger(i,Task(c, $\delta$ ),s)* $\rightarrow \delta >$ make $\delta$ to be executed. Action *Promises* can attach action *Inform*, for example, "coffee is coming".

*Refuse(i,j,Task(c, $\delta$ ))* denotes agent *i* refuse to perform the request $\delta$ from agent *j*, the precondition is that *Intend(i,Task(c, $\delta$ ),s)* is not hold. Action *Refuse* can attach action *Inform*, for example, "sorry, I am busy".

*Result(i,j,Task(c, $\delta$ ))* denotes agent *i* report the result to agent *j* about performing the task $\delta$. When agent *i* has finished the task $\delta$, action *Result* makes the fluent *Trigger* be not hold, *Result* action can also attach action *Inform*, for example, "Here is your coffee".

## 4. Cooperation Model and Semantics

Based on the communication actions above as mentioned, we bring forward a Requirement/Service cooperation model in multi-agent system (Fig.1).

In the model, there are two types of agents: request agents (RequAgtSet) and service agents (ServAgtSet). Request agents can request service agents to achieve a task, and service agents are able to correspondingly provide request agents with service in a cooperative way among service agents. That is to say, once a request agent ($RequAgt_i$) sends his request to a service agent, e.g., $ServAgt_1$, then $ServAgt_1$ is responsible for coordinating

among service agents for achieving the request. Firstly, $ServAgt_1$ estimates whether the request can convert to own task or not. If not, he sends this request to the second agent $ServAgt_2$ who replays his promise or refusal; if $ServAgt_1$ gets the refusal from $ServAgt_2$, then he sends the request again until a certain agent $ServAgt_i$ accepts request, or he will inform $RequAgt_i$ of not being able to serve the request. If an agent $ServAgt_i$ accepts the request, she will perform corresponding actions to achieve the task.

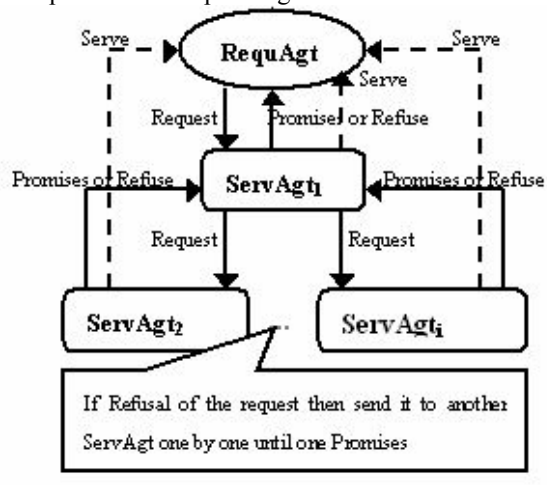


Fig.1. Requirement/Service cooperation model

Inspired by Singh's work [9], under the framework of Situation Calculus, we explain semantics of communication actions as satisfied conditions of performing communication actions. It is called Whole-hearted Satisfaction Semantics.

In order to denote that the communication action *Act* is Whole-heartedly Satisfaction, we introduce W operator: $W(Act,s)$, and $\exists s1.Do(Act,s,s1)$ means *Act* is successfully performed. Requirement/Service Cooperation model's semantics is formalized as follows:

$$W(\mathrm{Re}quest(i,j,Task(c,\delta)),s) \equiv \exists s1,s^*.Do(\mathrm{Re}quest(i,j,Task(c,\delta)),s,s1) \wedge s1<s^* \wedge W(\mathrm{Pr}omises(j,i,Task(c,\delta)),s^*). \quad (2)$$

In formula (2), action *Request* (from agent *i* to agent *j*) is Whole-heartedly Satisfaction iff agent *i* performs action *Request* to agent *j*, and action *Promises* (from agent *j* to agent *i*) is Whole-heartedly Satisfaction.

$$W(\mathrm{Pr}omises(j,i,Task(c,\delta)),s) \equiv [\exists s1,s^*.s<s1 \wedge Intend(j,Task(c,\delta),s1) \wedge s1<s^* \wedge Do(\mathrm{Pr}omises(j,i,Task(c,\delta)),s1,s^*) \wedge W(\mathrm{Re}sult(j,i,Task(c,\delta)),s^*)] \vee [\exists s1,s^*.s<s1 \wedge \neg Intend(j,Task(c,\delta),s1) \wedge (i \in \mathrm{Re}quAgtSet) \wedge s1<s^* \wedge \exists k.k \in ServAgtSet \wedge W(\mathrm{Re}quest(j,k,Task(c,\delta)),s^*)]. \quad (3)$$

Formula (3) means, it holds that action *Promises* (from agent *j* to agent *i*) is Whole-heartedly Satisfaction iff either it holds that the request of agent *i* can convert to the task of agent *j* (i.e., *Intend(j,Task(c, δ),s1)*), and agent *j* performs the action *Promises* to agent *i*, and action *Result* (from agent *j* to agent *i*) is Whole-heartedly Satisfaction; Or the request of agent *i* can not convert to the task of agent *j* (i.e., ¬*Intend(j,Task(c, δ),s1)*) and requester *i* belongs to RequAgtSet, agent *j* looks for an agent *k* in the service agents and transmit the request, and action *Request* (from agent *j* to agent *k*) is Whole-heartedly Satisfaction. Notice, since the task can be described using complex actions (or procedure) in extended ConGolog, we assume that the agents have always abilities to achieve her task.

$$W(\mathrm{Re}sult(j,i,Task(c,\delta)),s) \equiv \exists s1,s2.Do(\delta,s,s1) \wedge s1<s2 \wedge Do(\mathrm{Re}sult(i,j,Task(c,\delta)),s1,s2). \quad (4)$$

In formula (4), action *Result* (from agent *j* to agent *i*) is Whole-heartedly Satisfaction iff agent *j* has achieved task $\delta$, and agent *j* performs action *Result* to agent *i*.

## 5. Requirement/Service Cooperation Case of Delivering Coffee

According to our Requirement/Service cooperation model and semantics, we give a case of delivering coffee. Given a coffee bar scenario, the agents consist of two sets: customer agents (CAgtSet) and service agents (SAgtSet), whose relationship is the same as the model above.

The procedures *W_Request, W_Promises* and *W_Result* are the implementation of the above cooperation semantics. The symbol $\delta$ stands for a ConGolog Procedure, e.g., *deliCoff*. The procedure *deliCoff* (denotes agent *sagt* delivers a cup of coffee to agent *cagt*) is adapted from [4], including three primitive actions, i.e., *pickup(sagt,* Coff*)*, *give(sagt, cagt,* Coff*)*, *goto(sagt, Loc)*, two fluents (*Holding*, *Location*) and two special fluents (*Intend*, *Trigger*). The control procedure describe the coffee bar scenario in which there are three customers (C1, C2, C3) and two waiters (S1, S2), and customers can request any a waiter time after time. Main code of the procedures is as follow.

```
// Requirement/Service cooperation
Proc W_Request(i, j, Task(c, δ ))
   Request(i, j, Task(c, δ ));
   W_Promises(j, i, Task(c, δ ));
EndProc
Proc W_Promises(j, i, Task(c, δ ))
  if Intend(j, Task(c, δ )) then
```

```
      (Promises(j, i, Task(c, δ ));
      W_Result(j, i, Task(c, δ ));)
  else
    (if i∈CAgtSet then
        (AskSAgtSet={j}∪{};
        While ((¬∃k. k∈AskSAgtSet ∧ Intend(k,Task(c,
δ )))∧(∃m. m∈SAgtSet ∧ m∉ AskSAgtSet)) do
          AskSAgtSet={m}∪AskSAgtSet;
          W_Request(j, m, Task(c, δ ))
        EndWhile;
        if ¬∃k. k∈SAgtSet ∧ Intend(k,Task(c, δ )) then
          Refuse(j, i, Task(c, δ ));)
    else
        Refuse(j, i, Task(c, δ ));)
EndProc
Proc W_Result(j, i, Task(c, δ ))
   δ ; Result(j, i, Task(c, δ ));
EndProc

//Deliver Coffee
Proc deliCoff(sagt,cagt)
   if location(sagt)≠loc(CM) then
      goto(sagt, loc(CM))
   else
      (pickup(sagt,Coff);
      goto(sagt, loc(cagt));
       give(sagt,cagt,Coff))
EndProc

// Control Procedure
((π sagt. W_Request (C1, sagt, Task(C1,deliCoff)))*||
(π sagt. W_Request (C2, sagt, Task(C2,deliCoff)))*||
(π sagt. W_Request (C3, sagt, Task(C3,deliCoff)))*)
   >>
(<trigger(S1,Task(cagt,deliCoff))→deliCoff(S1, cagt)>||
<trigger(S2,Task(cagt,deliCoff))→deliCoff(S2, cagt)>)
```

## 6. Discussion

As for our Requirement/Service cooperation model, if the communication way in which a service agent who is responsible for coordinating sends request is substituted by the broadcast, correspondingly our model and semantics can be changed easily to a similar Contract Net model and semantics.

In fact, the Requirement/Service cooperation can be distinguished as two types: Terminating and Non-terminating. As for the former, as long as the request has been served, the Requirement/Service cooperation relation will terminates. For instance, "Please give me a cup of coffee". Actually, our Model put forward above is regarded as the Terminating Requirement/Service cooperation model. As for the later, after the request is occurred, service will be provided repeatedly as long as the condition is satisfied. For example, "Please deliver me my mails whenever they arrive". In terms of the interrupt mechanism $\langle \phi \rightarrow \delta \rangle$ in ConGolog, our model and semantics can be easily modified to the Non-terminating Requirement/Service cooperation model and semantics.

**Acknowledgments**

This work was based on the Planning Project supported by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (No. 03kjd520175) and the Project supported by the Social Development Plan of Jiangsu province, China (No. BS2001046).

## References

[1] A. Haddadi, *Communcaition and Cooperation in Agent Systems*, New York: Springer-Verlag, 1996.

[2] M. N. Huhns, L.M. Stephens, *Multiagent System and Societies of Agents. In: Multiagent System —A Modern Approach to Distributed Artificial Intelligence*, MA: MIT Press, 1999.

[3] Wang Huaimin, Wu Quanyuan, "A Formal Framework of Multi-agent Systems with Requirement/Service Cooperative Style", *Journal of Computer Science and Technology*, 2000, 15(2), pp. 106-115.

[4] Raymond Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MA: MIT Press, 2001.

[5] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl, "GOLOG: A logic programming language for dynamic domains", *Journal of Logic Programming*, 1997, 31, pp. 59-84.

[6] G. D. Giacomo, Y. Lespérance, H. Levesque, "ConGolog, A Concurrent Programming Language Based on the Situation Calculus", *Artificial Intelligence*, 2000, 121(1-2), pp. 109-169.

[7] R. Scherl and H. J. Levesque, "Knowledge, Action, and the Frame Problem", *Artificial Intelligence*, 2003, 144(1-2), pp. 1-39.

[8] S. Shapiro, Y. Lespérance, and H. J. Levesque, *The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems*, Proc. of the First Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02), ACM Press, 2002, pp. 19-26.

[9] Munindar P. Singh, *Multiagent System*: *A Theoretical Framework for Intentions, Know-how, and Communications,* New York: Springer-Verlag, 1994.

**Liu Yisong** is an Associate Professor in the School of Computer Science and

Telecommunication Engineering, Jiangsu University, China. He is working towards his Ph.D degree in the School of Computer Science and Technology at the Nanjing University of Science & Technology, China. His research interests include Virtual Reality, Intelligent Virtual Agent and Reasoning about Action.