ISSN 1009-3095 Journal of Zhejiang University(SCIENCE) V.2,No.2,P.132-136,Apr.-June, 2001
http://www.chinainfo.gov.cn/periodical; http://www.zju.edu.cn;
http://lib.zju.edu.cn/English; http://www.zjupress.com

ANALYSIS OF THE KQML MODEL IN MULTI-AGENT INTERACTION *

LIU Hai-long(刘海龙), WU Tie-jun(吴铁军)

(*National Laboratory for Industrial Process Control Technology, Institute of Intelligent Systems and Decision Making, Zhejiang University, Hangzhou 310027, China*)

Received June 18, 1999; revision accepted Apr.18,2000

Abstract: Our analysis of the KQML(Knowledge Query and Manipulation Language) model yielded some conclusions on the knowledge level of communication in agent-oriented program. First, the agent state and transition model were given for analyzing the necessary conditions for interaction with the synchronal and asynchronous KQML model respectively. Second, we analyzed the deadlock and starvation problems in the KQML communication, and gave the solution. At last, the advantages and disadvantages of the synchronal and asynchronous KQML model were listed respectively, and the choosing principle was given.

Key words: multi-agent, KQML, communication
Document code: A **CLC number:** TP18

INTRODUCTION

In the multi-agent theory, there are two communication models dealing with knowledge sharing and problem distributed solving:

1. Public storage, communication among agents can be achieved by building up public storage area, which every agent can visit. Black board model is the typical application (Stefan, 1994).

2. Message transfer, the information is exchanged among some agents directly with surroundings support. Agent communication language ACL (Michael, 1994; Thomas, 1992) is a representation. In addition, the AOP language (Shoham, 1993) was raised for agent programming, presented a meta-language for agent communication. It also belongs to message transfer model. There are some difficulties in establishing public storage in heterogeneous and distributed multi-agent system. So the research on a message transfer model becomes a focus in multi-agent communication.

Mauro (1994) did some primary research in support of agent oriented programming on label level. He gave the synchronal and asynchronous communication model of KQML, and showed the problems of resources starvation and deadlock in synchronal communication. But he did not analyze the reason and give the solution. In this paper, we established the agent state model and the KQML transition model, analyzed all the above problems, and gave the solution.

AGENT STATE MODEL AND KQML TRANSITION MODEL

Synchronal and asynchronous communication models are two ways for message transfer. There is no buffer in the synchronal communication, and the acceptor should stay in waiting state before receiving the message. On the other hand, there is a buffer in asynchronous communication, so the message can be received when the acceptor is idle. The set of KQML instructions is too large. For convenience, the instruction subset consisting of "assert", "tell" and "ask" are listed. We can explain how to implement synchronal and asynchronous communication in KQML.

1. Agent state model space

If A is an agents set, P is an agent working processes set, STATES:: = $\{$IDLE, BUSY, WAIT(a) $|$ $a \in A\}$ is an agent state set. Then we can define agent state model space.

Definition 1. Agent state model space is defined as three-tuple $\Gamma = \{\langle a, p, \text{state}\rangle \mid a \in$

* Project supported by the National High-Tech R&D Plan (9845-005)

A, $p \in P$, state $\in$ STATES$\}$. $\forall\ \gamma \in \Gamma$, is called agent state model.

If Γ_{KQML} is denoted as agent state model space of KQML, we can describe Γ_{KQML} as:

$\Gamma_{\text{KQML}}::=\Gamma_{\text{IDLE}}\oplus\Gamma_{\text{BUSY}}\oplus\Gamma_{\text{WAIT}}$
$\Gamma_{\text{IDLE}}::=\{\langle a\rangle \mid a\in A\}$
$\Gamma_{\text{BUSY}}::=\{\langle a, p\rangle \mid a\in A, p\in P\}$
$\Gamma_{\text{WAIT}}::=\{\langle a, p, \text{wait}(a')\mid a\in A, a'\in A\rangle\}$
"$\oplus$" is an operator for constructing state model spaces.

2 KQML state transition model

Definition 2. KQML transition model can be expressed as a three-tuple(Γ_{KQML}, L, $\{\rightarrow^{l} \mid l\in L\}$)(Mauro, 1994), in which, L is an instruction set; $\rightarrow^{l}: \Gamma_{\text{KQML}}\rightarrow\Gamma_{\text{KQML}}$ is transition relation satisfying a set of axioms and inference rules for $\forall\ l\in L$.

For different axioms and rules system, we can construct different KQML transition models. a1 – a4 and b1 – b5 give two sets of rules for point to point communication in KQML model respectively. In a1 – a4, L = {assert (a), tell (a, a'), ask(a, a'), $\surd$}, $\Gamma\subseteq\Gamma_{\text{KQML}}=\Gamma_{\text{IDLE}}\oplus\Gamma_{\text{BUSY}}\oplus\Gamma_{\text{WAIT}}$, and function "dispatch" maps the message is received by the agent to the corresponding process.

a1) $\langle a, \text{assert}(a').P\rangle\oplus\Gamma\oplus\langle a'\rangle \rightarrow^{\text{assert}(a',a)}\langle a, P\rangle\oplus\Gamma\oplus\langle a', P'\rangle$
where P' = dispatch (assert(a'))

a2) $\langle a, \text{ask}(a').P\rangle\oplus\Gamma\oplus\langle a'\rangle\rightarrow^{\text{ask}(a',a)}\langle a, P, \text{wait}(a')\rangle\oplus\Gamma\oplus\langle a', P'\rangle$
where P' = dispatch (ask(a', a))

a3) $\langle a, \text{tell}(a').P\rangle\oplus\Gamma\oplus\langle a', P', \text{wait}(a)\rangle\rightarrow^{\text{tell}(a',a)}\langle a, P\rangle\oplus\Gamma\oplus\langle a', P''\rangle$
where P' = dispatch (tell(a'), P')

a4) $\langle a,\surd\rangle\oplus\Gamma\rightarrow^{\surd}\langle a\rangle\oplus\Gamma$

From rule a1 and a2, we draw the conclusion that the message of "assert" and "ask" can be received by the acceptor without delay, because the acceptor is in IDLE. From rule a2 and a3, we know one side is always in waiting state after sending out "ask" message. So the "tell" message can also be received without buffer. There are no message saving and loading processes with buffer, and the sending and receiving message is a whole block at a time. So we have the following conclusions:

Conclusion 1. The communication in KQML state model satisfying the transition rules of a1 – a4 is a synchronal process.

In the communication model corresponding to b1 – b5, L = {assert (a), tell (a, a'), ask (a, a'), get(a), $\surd$}. $m(\cdot,\cdot)$ denotes the state model of the buffer.

"+" denotes the combining of agent state model and buffer state model.

b1) $\langle a, \text{assert}(a').P\rangle\oplus TS\rightarrow^{\text{assert}(a',a)}\langle a, P\rangle\oplus TS + \{m(a', \text{assert}(a'))\}$

b2) $\langle a, \text{ask}(a').P\rangle\oplus TS\rightarrow^{\text{ask}(a',a)}\langle a, P\rangle\oplus TS + \{m(a', \text{ask}(a', a))\}$

b3) $\langle a, \text{tell}(a').P\rangle\oplus TS\rightarrow^{\text{tell}(a',a)}\langle a, P\rangle\oplus TS + \{m(a', \text{tell}(a'))\}$

b4) $\langle a,\surd\rangle\oplus TS\rightarrow^{\surd}\langle a\rangle\oplus TS$

b5) $\langle a\rangle\oplus TS + \{m(a, \text{msg})\}\rightarrow^{\text{get}(a)}\langle a, p'\rangle\oplus TS$
where p' = dispatch(msg)

According to rules b1, b2 and b3, agent a can perform instructions "assert", "ask" and "tell" without delay. And he does not need to judge which state the action object is in. b5 is the rule that agent a receives information flow of the system. Under this rule, agent a need to be in the "IDLE" state, and the information flow m (a, msg) need to be in the buffer. There is no state judgement in the communication process under this rule set. So we can draw the following conclusion:

Conclusion 2. The communication in KQML model satisfying the transition rules of b1 – b5 is an asynchronous process.

3. The trace of state transition

Definition 3. Supposing the initial state of multi-agent system A is "IDLE", namely $\gamma_0\in\Gamma_{\text{IDLE}}$. The succeeding transition processes starting from the instruction χ_0 is:

$\gamma_0\rightarrow^{\chi_0}\gamma_1\rightarrow^{\chi_1}\gamma_2\cdots\rightarrow^{\chi_n}\gamma_{n+1}\cdots$

Then the trace of state transition is instructions array $\{\chi_0, \chi_1, \cdots, \chi_n\cdots\}$, denoted as $T(\gamma_0)_{\chi_0}$.

In addition, we use $t_i\in T(\gamma_0)_{\chi_0}$ to denote the ith term of the trace $T(\gamma_0)_{\chi_0}$, and $t\downarrow a$ denotes the instructions array only agent a participates in. There are two situations for $t\downarrow a$: (i) an infinite instructions array of communication; (ii) a finite instructions array ending with

a finished process.

ANALYSIS OF KQML MODEL

Before analyzing the features of the KQML model, we give some definitions of the KQML model.

1. Instruction enable space

Definition 4. Supposing $\xi \subset \Gamma_{KQML}$ is a sub-space of KQML state model space, $\chi \in L$ is an instruction in instruction set L. When the rule corresponding to instruction χ is enabled in ξ, that is to say $\exists\, \xi' \in \Gamma_{KQML}$, and $\xi \rightarrow^{\chi} \xi'$ holds, then we call the transition of instruction χ as enabled in sub-space of state model, denoted as: $\xi \vdash \chi$, and also we call ξ a sub-space of enabled state model of χ, or enabled space for short.

In addition, the set consisting of all the enabled space of χ is called the enabled space set, denoted as $\rho(\chi)$; the set that is the mapping of all the enabled spaces in $\rho(\chi)$ is called value space, denoted as $R(\chi)$.

After acting on ξ, the enabled instruction χ in an enabled space ξ will make χ change into state space ξ', so the instruction χ can be viewed as a function of state model space, denoted as $\chi : \Gamma_{KQML} \rightarrow \Gamma_{KQML}$. The definition field of function χ is $\rho(\chi)$, its value field is $R(\chi)$, and they satisfy $\rho(\chi) \subseteq \Gamma_{KQML}$, $R(\chi) \subseteq \Gamma_{KQML}$.

Definition 5. Supposing ξ is enabled space of function χ. If a is an agent that can execute instruction χ in enabled space, then we say a can activate χ in space ξ.

"Enabled" is used to describe the space that χ can act on, while "activation" is used to describe the relation between agent and instruction in enabled space. The following example will explain the situation above:

Example 1. Supposing three agents a, b, $c \in A$, their state models are $\gamma_a = \langle a, \text{assert}(b).p_a \rangle$, $\gamma_b = \langle b \rangle$, $\gamma_c = \langle c, p_c \rangle$, and $\xi = \{\gamma_a, \gamma_b, \gamma_c\}$ respectively. If we use synchronal transition rules of KQML, the rule condition part corresponding to $\chi = \text{assert}(b)$ is satisfied in this sub-space of state model. We also can find a sub-space of state model $\xi' = \{\langle a, p_a \rangle, \gamma_b = \langle b, pb' \rangle, \langle c, p_c \rangle\}$, in which $\xi \rightarrow^{\text{assert}(b,a)} \xi'$ holds. Then the rule assert (a', a) corresponding to instruction χ is enabled in ξ. In ξ, if agent a can observe the state of b is IDLE, then a can trigger instruction χ. So we say a can activate χ in the space ξ.

2. Feature of synchronal KQML

In example 1, if agent a cannot observe the state of b, a will not trigger instruction χ. Although a is in the enabled space of χ, a cannot activate χ. So, in agent communication, agent a must satisfy two conditions before sending communication instruction χ:

1) agent a is in the enabled state space of χ;

2) agent a can activate χ in the enabled space of χ.

In the synchronal communication model of KQML, the activation is based on the observing state of the agent on which the instruction acts. So state observing is an essential condition for the synchronal KQML model. In fact, to judge the activation is to judge if its state space is enabled.

Conclusion 3. Supposing A is an agent set, Γ_A is the state model space, L is a communication instructions set. Then the necessary condition of the synchronal model of KQML is: $\forall \chi \in L, \forall a \in A$

$$\Gamma_A \in \rho(\chi) \Leftrightarrow \kappa_a \Gamma_A \in \rho(\chi)$$

here, "κ_a" is a know operator of normal modal logic, $\kappa_a \Gamma_A \in \rho(\chi)$ means agent a knows the proposition $\Gamma_A \in \rho(\chi)$ holds.

In conclusion 3, proposition "$\Gamma_A \in \rho(\chi) \Rightarrow \kappa_a \Gamma_A \in \rho(\chi)$" means agent a knows that the state model space he is in, is enabled space of χ. So it ensures the activation of χ. Proposition "$\kappa_a \Gamma_A \in \rho(\chi) \Rightarrow \Gamma_A \in \rho(\chi)$" means agent a is truly in state model space $\rho(\chi)$. So it ensures the validity of the activation of χ.

3. Feature of asynchronous KQML

In the asynchronous KQML communication model, supposing $\forall \chi \{\text{assert, tell, ask}\}$, the enabled space $\rho(\chi)$ of χ is:

$$\rho(\chi) = \{\xi \mid \xi \vdash \chi\}$$

In b1 - b5, $\forall a \in A$, suppose the state model space which a is in, is Γ_{KQML}, then the sufficient and necessary condition for proposition "$\Gamma_{KQML} \in \rho(\chi)$" is:

$\exists a' \in A$, and $a \neq a' \langle a, \chi(a').P \rangle \in \Gamma_{KQML}$

So, for agent a and $\forall \chi \in \{$assert, tell, ask$\}$, the activation of χ is only related to its own state. They are whether the next program is $\chi(a')$ and $a \neq a'$. The reason to limit $a \neq a'$ is to prevent agent a from activating the nonsensical instruction to establish a communication link with himself.

The asynchronous KQML communication model includes instruction "get" to load message flow in the buffer. This is the key distinction between asynchronous and synchronous communication. In b1) – b5), for $\forall a \in A$, the state model space which a is in, is Γ_{KQML}. Then the sufficient and necessary condition of proposition $\Gamma_{KQML} \in \rho(\text{get}(a))$ is:

$\langle a \rangle + \{m\ (a, \text{msg})\} \in \Gamma_{KQML}$

So, the activation of instruction "get(a)" only applies to the state of agent a and buffer.

DEADLOCK AND RESOURCE STARVATION

Mauro pointed out that there are resource starvation and deadlock in synchronal KQML communication.

1. Resource starvation

Resource starvation arises when one activated process monopolizes the necessary resource enduringly, so that other processes cannot be activated in finite time. For example, two agents b and d, try to send "assert" and "ask" instruction to the same agent a:

Suppose: B = assert(a).B
D = ask(a).D
Dispatch(ask (a, d)) = tell(d).$\surd$
Dispatch(assert(a, b)) = $\surd$

And suppose state space is $\{\langle d, D \rangle, \langle b, B \rangle, \langle a \rangle\}$, then instructions "assert(a, b)" and "ask(a, b)" are enabled:

$\{\langle d, D \rangle, \langle b, B \rangle, \langle a \rangle\} \rightarrow^{\text{assert}(a, b)} \{\langle d, D \rangle, \langle b, B \rangle, \langle a, \surd \rangle\}$
$\{\langle d, D \rangle, \langle b, B \rangle, \langle a \rangle\} \rightarrow^{\text{ask}(a, d)} \{\langle d, D, \text{wait}(a) \rangle, \langle b, B \rangle, \langle a, \text{tell}(d).\surd \rangle\} \cdots$

If ask(a, d) is activated, then we will get an infinite nested process, and its state transition trace $t \downarrow d$ is an infinite instruction array {ask (a, d), ask(a, d), $\cdots$}. So it leads to resource starvation.

The essential reason for resource starvation is that when a process monopolizes some resources (for example, monopolize CPU, communication port and link), it leads to infinite loop or infinite nested process. So resource starvation is not related to communication model (synchronal or asynchronous). Process monopolizing resource should be avoided in agent-oriented programming.

2. Deadlock

Deadlock occurs when an agent a wants to create a communication link of instruction χ, but in finite time, the state model space Γ, which a is in, is not enabled space, namely $\Gamma \notin \rho(\chi)$. There are two situations in deadlock: one is that both of two agents want to be communication sponsor and create communication link with the other concurrently, but none of them will recede. So the establishment of communication link fails; the other situation is that agent a is in an infinite loop process, this makes the other agent, which wants to create a communication link with a, stay in waiting state.

Lacking overall control and recede coordination in communication, agents are in mutual waiting state when they both want at the same time to create a communication link. This leads to the first situation. It is the general problem in synchronous communication when inquire instruction (for example "ask") transits by block without buffer (Mauro, 1994). So in communication based on label level, establishing an effective control and recede coordination mechanism can ensure that synchronous communication works well.

From conclusion 2, the necessary condition for establishing a KQML model is that the agent has the ability to judge the enabled space. The key of judging enabled space is to get the state of its own and related agents. For example, the activated state model of rule "assert(b, a)" are $\langle a$, assert (b).$P \rangle$ and $\langle b \rangle$ for a, b respectively. That is to say the next instruction of a is assert(b), and the state of b is IDLE. At communication level, the agent needs to get the state of related agents. The agent can also find out if he is in deadlock state while he judges if the enabled space exists. For example, the state of agents a and b are: $\langle a$, assert (b).$P_a \rangle$ and

$\langle b, \text{ask}(a).P_b\rangle$ respectively. When a and b prepare to execute assert(b, a) and ask(a, b) at the same time, if the state can be known by each other, then a and b will know that they are in deadlock state. This needs mutual coordination and receding. How to coordinate and recede is another problem in the multi-agent research field. This paper will not deal with this situation.

The second situation of deadlock is just like that of resources starvation, because an infinite loop and an infinite nest process will lead to deadlock. We cannot solve it at communication level. But we can try to avoid it in the agent program. Generally, we use a static limiting method to avoid deadlock. For example, prohibit agent from sending inquire that may cause infinite loop calculation to other agents; prohibit mutual recursion request and so on.

In asynchronous communication, establishing a communication link does not need to wait for a particular enabled state of the other agent. So deadlock will not occur. The following can explain why this is so:

Supposing the state model space of system is Γ in sometime, the state model of agent a is $\langle a, \chi.P\rangle$, and $\chi \in \{\text{ask, assert, tell}\}$. In b1 – b5, we know $\Gamma \in \rho(\chi)$ holds. So agent a can activate instruction χ, and deadlock does not occur in asynchronous communication model based on b1) – b5).

CONCLUSIONS

From Conclusion 2, there must be a mechanism, which can give agents messages about their state, to implement KQML synchronal communication. So establishing KQML synchronal communication is more difficult than KQML asynchronous communication.

Message transmission by asynchronous communication can support higher knowledge level program (Angela, 1995; Newell, 1982). But in the distributed problem solving, asynchronous communication cannot use the last results to execute its next solving process in time. That reduces the efficiency of distributed solving. Analyzing the communication mechanism, we know synchronism communication has no such problem.

These two communication models have their own advantages. Which one should be chosen depends on the practical objects. Generally, for knowledge sharing problem, it is not necessary to get knowledge in real time, so we can choose the asynchronous communication model for convenience. But for distributed problem solving, we should consider adopting the synchronal communication model.

References

Angela Dalmonte, Mauro Gaspari, 1995. Modelling interaction in agent system, technical report UBLCS-95-7, laboratory for computer science, University of Bologna.

Mauro Gapari, Enrico Motta, 1994. Symbol-level Pequirment for agent-level rrogramming, technical report UBLCS-94-2, Laboratory for Computer Science, University of Bologna.

Michael, R. Genesereth, Steven, P. Ketchpel, 1994. Software Agents. *Communication of ACM*, **37**: 48-54.

Newell, A. 1982. The knowledge level. *Artificial Intelligence*, **18**: 87 – 127.

Shoham, Y., 1993. Agent-oriented programming. *Artificial Intelligence*, **60**: 51 – 92.

Stefan, K., 1994. STRICT: A blackboard based tool supporting the design of distributed PPC application. *Expert Systems With Applications*, **7**(1): 10 – 20.

Thomas, R. Gruber, 1992. A translation approach to portable ontology specifications. Computer Science Department, Stanford University Tech Report KSL 92 – 71.