멀티-에이전트 시스템 협상을 위한 논리적인 에이전트 통신 언어에 관한 연구 (I)

Research on a Logical Agent Communication Language for Multi-Agent Systems Negotiation (I)

이 명 진* 한 현 관**
Myung-Jin Lee Hyun-kwan Han

요 약

멀티-에이전트 시스템에서 에이전트는 다른 에이전트와 협상하기 위하여 공동의 에이전트 통신 언어를 사용하여야 하고, 일치에 이르도록 설계된 협상 프로토콜에 의해 행동하여야 한다. 따라서 에이전트는 위의 요구 사항들을 수용할 수 있는 적당한 아키텍쳐를 가져야 한다. 이 논문에서 우리는 유익한 에이전트 통신 언어를 정의하고, 다른 에이전트 통신 언어들(가령, FIPA(Foundation for Intelligent Physical Agents) 에이전트 통신 언어와 KQML(Knowledge Query Manipulation Language))과 여기서 소개한 에이전트 통신 언어를 비교한다. 특히 여기서는 에이전트를 지식베이스와 협상 라이브러리를 가진 논리 프로그램으로 표현한다. 마지막으로 협상 라이브러리 안에 있는 계획자(planner)가 어떻게 행위들의 계획을 제공하고, 어떻게 에이전트의 지식베이스를 갱신하는가를 보인다.

Abstract

Agents in Multi-Agent Systems (MAS) should make use of a common Agent Communication Language (ACL) in order to negotiate with others, and conform to negotiation protocols thatare designed to reach agreements. Therefore, agents must have suitable architectures that could cover above requirements. In this paper, we define an instructive ACL and compare it with other ACLs such as Foundation for Intelligent Physical Agents (FIPA) ACL and Knowledge Query Manipulation Language(KQML). In particular, we represent agents as logic programs with knowledge base and negotiation library. Finally, we show how the planner, which is in the negotiation library, provides the plan of actions and updates agent's knowledge base.

☞ Keyword : Logic Programming, Logic Agent, Agent Communication Language, Negotiation.

1. Introduction

The main objective of an ACL is to model a suitable framework that allows heterogeneous agents to interact and to communicate with meaningful statements that convey information about their environments or knowledge[3]. A good ACL should be declarative, syntactically simple, and readable by people. It should be concise, yet easy to parse and to generate[4]. To transmit a statement of the language to another agent, the statement must be passed through the bit stream of the underlying transport mechanism. Thus, the language should be linear or should be easily translated into a linear form. Finally, because a communication language will be integrated into a wide variety of systems, its syntax should be extensible.

Negotiation plays a central role in multi-agent applications. In MAS, agents interact in order to exchange knowledge, when they are not enough to

* 정 회 원 : 아시아대학교 IT Master학과 교수
mjleekor@korea.com, hanhyoun@tpic.ac.kr
** 정 회 원 : 대구산업정보대학 컴퓨터정보계열 겸임교수
hanhyoun@tpic.ac.kr
[2006/06/01 투고 - 2006/06/21 심사 - 2006/10/09 심사완료]

achieve their goals. In domains such as e-commerce, when it is natural to assume that agents are self-interested, it is most likely that they need to negotiate in order to obtain required information or resources. The protocol used for negotiation has tobe properly designed in order to ensure that each party will have a positive payoff out of the negotiation process[13]. The negotiation protocol determines the flow of messages between the negotiating parties and is necessarily public and open. The negotiation strategy, on the other hand, is the way in which a given party acts within those rules in an effort to get the best outcome of the negotiation. For example, when and what to concede, and when to hold firm. Therefore, the negotiation strategy of each participant is necessarily private.

In this paper, we consider the following three modalities: *B*for beliefs used to represent agent's mental attitudes to the state of the environment, *D*for desires used to represent motivations of the agent, and *I* for intentions used to represent goals of the agent. We assume a multi-agent environment that exchanges the resources to achieve agents' goals. We represent agent's knowledge as declarative logic program, propose a simple ACL for negotiating with resource-bounded agents, compare our ACL with other ACLs such as FIPA ACL[5] and KQML[8], and compare our ACL with SQL and Prolog. Finally, we show how agent's planner provides the plan of actions and how the agent updates its own knowledge base.

2. ACL for Negotiation

The actual exchange of messages is driven by the participating agents' own needs, goals, or mental attitudes. Assumed agents are negotiating on the allocation of deficient resources, agents require the allocation of deficient resources to achieve their goals. In this case, we can simply define an ACL for resource-bounded agents as follows[9]:

ask_if(*a*1, *a*2, *m*) *inform*(*a*1, *a*2, *m*) *request*(*a*1, *a*2, *g*, *r*)
reject(*a*1, *a*2, *g*, *r*) *give*(*a*1, *a*2, *r*) *alternative*(*a*1, *a*2, *g*, [*subgoals*])
achieved_goal(*a*1, *a*2)

where *a*1 and *a*2 are agent's identifiers respectively, *m* is mental attitudes of the agent, *g* and *g* are agent's goals, *r* is resource, and *subgoals* is another plan for achieving the goal.

We now examine the relationships between our communicative acts, the KQML, and FIPA ACL. In the case of *request* act, the KQML has an *achieve* performative similar to *request* message. For example, agent *a*1 sends the following performative to agent *a*2, requesting that *a*2 set a new value for the motor *buick* of *park_avenue*_2000

```
(achieve
        :sender     a1
        :receiver   a2
        :language   Prolog
        :content    "buick(park_avenue_2000)" )
```

On the other hand, we consider a FIPA ACL *request* act that agent *a*1 requests *a*2 to give resource *r*

```
(request
        :sender     a1
        :receiver   a2
        :language   Prolog
        :content    "give(r)" )
```

Our act *request*(*a*1, *a*2, *g*, *r*) means that agent *a*1 requests deficient resource *r* from agent *a*2 to

achieve its goal g. However, there is an important difference between above KQML *achieve*, FIPA ACL *request*, and our *request*($a1$, $a2$, g, r). The KQML agent $a2$ does not know the reason why $a1$ sets the new value and the FIPA agent $a2$ does not know the reason why $a1$ requests resource r, while our agent $a2$ knows the reason why $a1$ requests resource r, i.e., because of the goal g. This fact helps an agent to reason about others' mental attitudes so that the agent may plan its goal more effectively. Because this information corresponds to *observe* in the *observe-think-act agent cycle* of Kowalski and Sadri[7], where changes in the environment (including communications between agents) can be observed as inputs, it could be used in Abductive Logic Programming (ALP) which performs the abductive/hypothetical reasoning about other agents[6, 13].

We also examine the relationships among our ACL, SQL, and Prolog. Mental attitude m could be represented as follows: $B(p)$, $D(p)$, or $I(p)$ where p is an atomic sentence. The use of *inform* for supplying new information to another agent is related to the data manipulation commands of SQL and Prolog. Assumed that the SQL database has belief, desire, and intention tables, called *b_table*, *d_table*, and *i_table*, respectively, SQL INSERT of a new row <p> into the belief table or Prolog *assert*(p) corresponds to sending an *inform*($a1$, $a2$, $B(p)$) message. Sending *inform*($a1$, $a2$, $B(p)$) with the negated sentence p corresponds to SQL DELETE of the respective row or Prolog *retract*. Table 1 shows these relationships among our ACL, SQL, and Prolog.

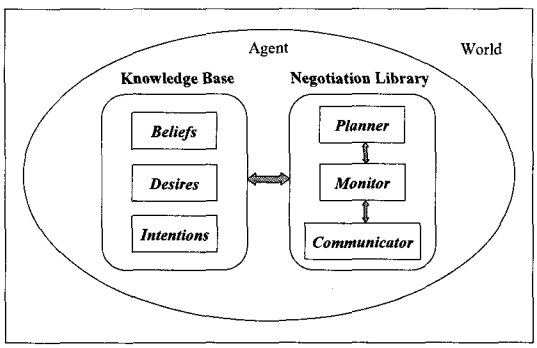
3. An Agent Architecture for Negotiation

Roughly speaking, mental attitudes of an agent, which represent abstract characteristics of the agent, can be described as follows: a set of beliefs about the world, a set of goals that the agent is currently trying to achieve, a library of plans describing how to achieve goals and how to react to changing in beliefs, and an intention structure describing how the agent is currently achieving its goals and reacting to changing in beliefs[10]. In general, an agent has a variety of knowledge to achieve its goals, to plan some tasks, and to communicate with other agents. For example, agents are required to possess knowledge that can be represented as a set of sentences. These sentences describe knowledge about their beliefs or capabilities, other interactive agents, how to communicate with others, and a specific application domain. Also, agents should have the capability of dealing with multi-interaction and communicating with others distributed by a network.

〈Table 1〉 The relationships among our ACL, SQL, and Prolog〉

Our ACL	SQL	Prolog
inform($a1$, $a2$, $B(p)$)	INSERT INTO *b_table* VALUES p	*assert*(p).
inform($a1$, $a2$, $B(p)$)	DELETE FROM *b_table* WHERE $x = p$	*retract*(p).
ask-if($a1$, $a2$, $B(p)$)	SELECT x FROM *b_table* WHERE $x = p$	?-p.
ask-if($a1$, $a2$, $B(p)$)	n.a.	?-*not* p.

In order to design such an agent, we consider a cooperative agent architecture containing the following two major components(<Fig. 1>): *knowledge base* and *negotiation library* including planner, monitor, and communicator. The knowledge base is a set of logical sentences, which includes knowledge about the agent's capabilities and other agents', and rules for problem decomposition. The factors in the knowledge base are represented by predicates, which mean mental attitudes of the agent. On the other hand, the negotiation library is responsible for deciding how to solve each task (*planner*), supervising the execution of tasks (*monitor*), and handling incoming and outgoing messages (*communicator*). Unlike blackboard systems, the communicator sends messages to other agents using the TCP/IP socket and redirects the received messages.



〈Fig. 1〉 A cooperative agent architecture

Negotiating agents share knowledge about actions. The planner provides not only a sequential plan of actions but also a list of deficient resources needed to achieve goals by approaching to the set of beliefs. On the other hand, when an agent receives an alternative/planwhich could achieve its goal, the planner chooses the lower cost plan comparing the original plan with the new plan. To do this, the planner uses the cost function, which maps the domain of possible plans to the number of deficient resources. The deficient resources are those the agent cannot acquire in its world.

When the planner receives *inform*(which tells truth/falsity of mental attitudes of other agents), *give* (which conveys resources), or *achieved_goal*(which tells the achievement of the goal) from other agents, it properly modifies its own mental attitudes*assertion* or *retraction*. For example, when *b*(*have*(*a*3, *hanger*)) is in *a*1's knowledge base, once *a*1 receives *give*(*a*3, *a*1, *hanger*) from *a*3 after *a*1 requests *hanger* from *a*3 to achieve its own goal, then *a*1 modifies its own beliefs and beliefs about *a*3.

Intelligent agents are software programs that use agent communication protocols to exchange information and to achieve theirconflicting goals and resources allocation. The interaction protocols for intelligent agents are agent communication rules and based on speech-act language theory. They can be used as negotiation protocols which specify the messages that each agent is allowed to make. In the real agents' communication messages, we apply a variant of KQML performatives and FIPA ACL acts to express the agent illocutionary forces.

Negotiation protocols could cover the permissible types of participants e.g., the negotiators and any relevant third parties, the negotiation states e.g., accepting requests or negotiation closed, the events which cause negotiation states to change e.g., no more requests or request accepted, and the valid actions of the participants in particular states e.g., which messages can be sent by whom, to whom, at what stage.

The process of negotiation starts when an agent generates a *request*message. Other agents then either

accept it, reject it, or make a counter-request. Following this, the originalagent then either sends clarifying information that may solve any problems, makes a new request, rejects the counter-request, or indicatesits acceptance of the counter-request. This process continues until all the agents involved agree on a request or they cannot reach an agreement.

4. Implementation

4.1 Environments

We show the possibility of application applying our ACL to a negotiation system of resource-bounded agents. The implementation is performed using InterProlog 2.0.1, which supports Java 2 SDK 1.4 andXSB Prolog 2.5. InterProlog is a programming environment for XSB Prolog and SWI Prolog. It consists of a Java application front-end that communicates with a Prolog system running as a subprocess, using standard console redirection and TCP/IP sockets. It is implemented as a set of standard Java classes and Prolog predicates.While there are some communication mechanisms such as stream-oriented and message-oriented, we use buffered, message-based communication mechanism using sockets: The communication process exchanges messages that have well-defined boundaries.

4.2 Axioms for Resource-bounded Agent

We follow the axiomatization of Rao as follows [12]:

- *goal-intention compatibility*: *intend() goal()*

  If an agent adopts as an intention then the agent should have adopted as a goal to be achieved.

- *intentions leading to actions: intend(do()) do()*

  If an agent has an intention to do particular action then the agent will do the action.

In addition to the axiomatization of Rao, we tries to confine intentions as much as possible, taking the action when the agent believes that it can take the action *intend(do()) can() do()*.

We consider the standard *KD*45 axiomatization for beliefs, Modus Ponens (MP) inference rule, and the following axioms for resource-bounded agents:

$b_i(have(X, Z)\ give(X, Y, Z))\ b_i(have(Y, Z))$. (4-1)

$b_j(have(j, Z)\ give(X, j, Z))\ b_j(have(j, Z))$. (4-2)

$b_i(have(X, Z)\ give(X, Y, Z))\ b_i(have(X, Z))$. (4-3)

$b_i(have(i, Z))\ b_i(holdon(i, Z))\ request(X, i,$
$give(i, X, Z))\ give(i, X, Z)$. (4-4)

$b_i(have(i, Z))\ b_i(holdon(i, Z))\ request(X, i,$
$give(i, X, Z))\ give(i, X, Z)$. (4-5)

$I_i(give(X, i, Z))\ request(i, X, give(X, i, Z))$. (4-6)

4.3 Knowledge bases of Agents

We also consider a variant of Parsons example[11]. Agent a1 tries to hang a picture, a2 a mirror, and a3 a clock. a1 needs a nail, a2 a hammer, and a3 a hanger_nail to achieve its goal. Now, each planner comes to know the intention of the agent, adopts the intention as a goal to be achieved, and decides an appropriate plan to solve the goal. For example, when the set of beliefs of a1 is as follows,

b(have, (a1, hammer)) *b(have, (a1, picture))*

b(have, (a1, screw)) *b(have, (a1, screwdriver))*

b(have, (a1, hanger_nail)) *b(have, (a3, hanger))*

the planner of *a*1 adopts *do*(*a*1, *hang_picture*) as a goal and decides a plan to achieve the goal using hammer, nail, and picture. The planner of *a*1 makes a list of deficient resources by searching required resources to perform the plan. In this case, the list will be [*nail*]. So, *a*1 asks *a*2 and *a*3 if they have a nail*ask_if*(*a*1, *a*2, *b*(*have*(*a*2, *nail*))) and *ask_if*(*a*1, *a*3, *b*(*have*(*a*3, *nail*))).

On the other hand, when *a*2 has the following set of beliefs, the planner of *a*2 decides a plan to achieve the goal using hammer, nail, and mirror.

b(*have*, (*a*2, *nail*)) *b*(*have*, (*a*2, *mirror*))

b(*have*, (*a*1, *screw*)) *b*(*have*, (*a*1, *hammer*))

b(*have*, (*a*1, *screwdriver*))

Because *a*2 has all resources to achieve its goal, the list of deficient resources will be empty. During the negotiation process, *a*2 comes to know another way to hang a mirror by receiving *alternative*(*a*1, *a*2, *hang_mirror*, [*screw*, *screwdriver*, *mirror*]). In this case, the planner of *a*2 adopts the lower resource cost plan by comparing the cost $|p|$ of the original plan p with the cost $|p|$ of the new plan p using the cost function. If we have the relation , the planner will adopt the new plan p. When an agent comes to know/belief the occurrence of (4-1), (4-2), or (4-3), the agent will properly modify its knowledge base.

The following list shows a part of consulting mental attitudes and negotiation mechanism for agents:

```
:- compiler_options([xpp_on]).
#include "socket_defs_xsb.h"
% Import necessary utilities
:- import member/2 from basics.
:- import load_dyn/1 from consult.
:- import socket/2, socket_bind/3, socket_listen/3,
   socket_accept/3, socket_set_option/3, socket_close/2,
   socket_recv/3, socket_send/3 from socket.

?- load_dyn(agent1).
?- reconsult(negotiation).
?- agent2.
?- agent3.
% Port on which Agent1 is listening
xsb_port1(6020).
xsb_port2(6022).
xsb_message1('You are Agent2').
xsb_message2('You are Agent3').
...
```

The following list shows a part of negotiation mechanism for agents:

```
:- import member/2 from basics.
:- import socket_close/2, socket_recv/3, socket_send/3
   from socket.
find_agent(b(have(Agent, Tool)), Agent2, Tool) :-
        b(Agent2, b(have(Agent2,Tool))).
find_goal(Task, Goal) :-
        rule(b(can(Agent, Goal)), Subgoals),
        member(Task, Subgoals).
plan(Task) :-
        Task.
plan(Task) :-
        find_agent(Task, Receiver, Tool),
        find_goal(Task, Goal),
        b(my_name(Sender)),
        send_message('request', Sender, Receiver,
        Goal, Tool),
        (Sender == 'a1' ->
          sid12(Sock12), wait_ reply(Sock12)
...
```

4.4 Negotiation Processes

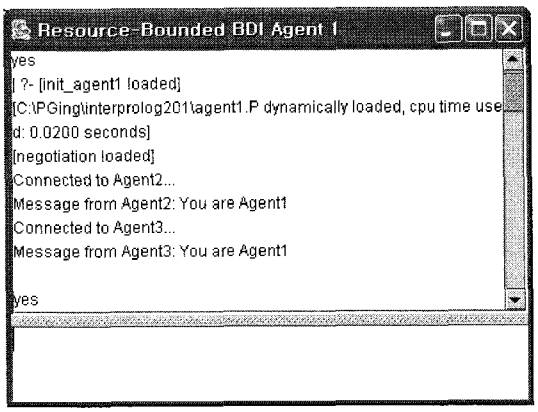
When we click *Start simple BDI agents negotiation* item, negotiation among agent *a*1, agent *a*2, and agent a3 starts. At first, *a*1 tries to achieve its goal, *intend*(*do*(*a*1, *hang_picture*)). Thus, it tries to solve the query, ?-*solve*(*b*(*do*(*a*1, *hang_picture*))). This query is transferred to the monitor of *a*1 and again passed on to the planner of *a*1. The planner decomposes the query and comes to know that *a*1 needs a nail. *a*1 however has no knowledge about nail so that it asks *a*2 and *a*3 if they have a nail, *ask_if*(*a*1, *a*2, *b*(*have*(*a*2, *nail*))) and *ask_if*(*a*1, *a*3, *b*(*have*(*a*3, *nail*))), and then it waits for a reply through the communicator.

On the other hand, *a*2 which receives an *ask_if* message from *a*1 checks its knowledge base, informs *a*1 that it has a nail, *inform*(*a*2, *a*1, *b*(*have*(*a*2, *nail*))), and waits for a reply. Now, *a*1 comes to know that *a*2 has a nail, requests a nail from *a*2, *request*(*a*1, *a*2, *hang_picture*, *nail*), and waits for a reply. This *request* message means that *a*1 requests a nail from *a*2 to achieve its goal, hanging a picture.

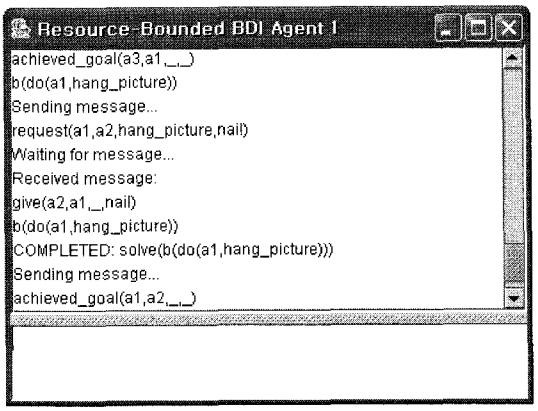
*a*2 which receives the *request* message from *a*1 first checks the unacceptable conditions of the resources in the message. It knows that the resources conflict occurs so that tries to achieve its goal, *intend*(*do*(*a*2, *hang_mirror*)). It decomposes the goal, but comes to know that it lacks a hammer to hang a mirror on its own. It requests a hammer from *a*1, *request*(*a*2, *a*1, *hang_mirror*, *hammer*), and waits for a reply.

*a*1 whichreceives a counter-request wants to know whether the resources conflict occurs. Unfortunately, the resources satisfies an unacceptable condition and it finds an alternative to achieving *a*2's goal and sends the alternative to *a*2, *alternative*(*a*1, *a*2, *hang_mirror*, *screwdriver*, *screw*, *mirror*), which represents that if *a*2 has a screwdriver, a screw, and a mirror then *a*2 can hang a mirror.

Finally, <Fig. 2> shows the state of *a*1 before negotiation and <Fig. 3> shows the state of *a*1 after negotiation.



〈Fig. 2〉 The state of a1 before negotiation



〈Fig. 3〉 The state of a1 after negotiation

5. Evaluation and Conclusion

In the paper, we represent resource-bounded agents as logic programs, define a simple ACL for negotiation, and show how the negotiation planner provides a plan of actions. Logic programs syntactically/declaratively represent knowledge of

agents and offer accurate semantics for communication acts. Unlike FIPA ACL and KQML, the agent receiving our *request* communication act knows the reason why the sending agent requests the resource. This fact will improve the reasoning ability of the receiving agent. In particular, ALP could perform the hypothetical reasoning using this fact.

While the planner, which makes a plan of actions, could have STRIPS-based, situation calculus-based, or event calculus-based planner, we have the planlibrary in the negotiation library. In the cooperative MAS examples, the planner decides a goal of the agent, sets up an appropriate plan to achieve the goal, and adopts the lower cost plan using the cost function. Although this research is performed in the restricted environment, we show that the planner takes adequate actions in the resource-bounded cooperative MAS.

The following issues require further investigation: research on dynamic protocols whichcan negotiate on the rules that will be used, research on ACLs and planners which can be used in the more open environments, and research on logic-based frameworks which can negotiate in one-to-many such as auction systems and e-commerce systems[1, 2].

References

[1] C. Bartolini and C. Preist. A Framework for Automated Negotiation. *HP Labs Technical Report* 2001-90, HP Labs Agent Research, 2001.

[2] C. Bartolini, C. Preist, and N. R. Jennings. A Generic Software Framework for Automated Negotiation. *HP Labs Technical Report*2002-2, HP Labs Agent Research, 2002.

[3] B. Chaib-draa and F. Dignum.Trends in Agent Communication Language. *Computational Intelligence*, 18(2), 2002.

[4] T. Finin, Y. Labrou, and J. Mayfield. Desiderata for Agent Communication Languages, In *Proceedings of the AAAI Symposium on Information Gathering from Heterogeneous Distributed Environments*, 1995.

[5] Foundation for Intelligent Physical Agents (FIPA). FIPA Communicative Act Library Specification. *http://www.fipa.org/specs/fipa*00037/*SC*00037*J.html*, 2002.

[6] A. C. Kakas, R. A. Kowalski, and F. Toni. The Role of Abduction in Logic Programming. *Handbook of Logic in AI and Logic Programming*, 5:235-324, 1998.

[7] R. A. Kowalski and F. Sadri. From Logic Programming to Multi-Agent Systems, *Annals of Mathematics and Artificial Intelligence*, 25:391-419, 1999.

[8] Y. Labrou and T. Finin. A Proposal for a New KQML Specification. *Technical Report CS*-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1997.

[9] M. J. Lee and J. S. Kim. A Logic Programming Framework for Negotiation among Resource-bounded BDI Agents. In *Proceedings of the International Conference on Intelligent Agents, Web Technologies, and Internet Commerce*, 2001.

[10] D. Morley. Semantics of BDI Agents and Their Environment. *Technical Note* 74, Australian Artificial Intelligence Institute, 1996.

[11] S. Parsons, C. Sierra, and N. R. Jennings. Agents that Reason and Negotiate by Arguing. *Journal of Logic and Computation*,

8(3):261-292, 1998.

[12] A. S. Rao and M. P. Georgeff. Intentions and Rational Commitment. *Technical Note* 8, Australian Artificial Intelligence Institute, 1993.

[13] F. Sadri, F. Toni, and P. Torroni. Logic Agents, Dialogues and Negotiation: An Abductive Approach. In *Proceedings of the Symposium on Information Agents for E-Commerce*, 2001.

◐ 저 자 소 개 ◑

이 명 진(Myung-Jin Lee)

1990년 대구대학교 수학과 졸업(학사)
1994년 계명대학교 대학원 컴퓨터공학과 졸업(석사)
2002년 계명대학교 대학원 컴퓨터공학과 졸업(박사)
2003년~현재 아시아대학교 IT Master학과 교수
관심분야 : 지능형 시스템, 전자상거래 시스템, 에이전트 시스템
E-mail : mjleekor@korea.com

한 현 관(Hyun-kwan Han)

1992년 경일대학교 전자정보공학과 졸업(학사)
2002년 대구대학교 대학원 산업정보학과 졸업(석사)
2007년 영남대학교 대학원 컴퓨터공학과 졸업(박사)
2004~현재 대구산업정보대학 컴퓨터정보계열 겸임교수
관심분야 : SE, UML, 컴포넌트
E-mail : hanhyoun@tpic.ac.kr