International Journal of Energy, Information and Communications
Vol.8, Issue 1 (2017), pp.39-50
http://dx.doi.org/10.14257/ijeic.2017.8.1.04

Design and Implementation of Agent Environment for Developing Nash-Q Learning Agents

Takahiro Uchiya[1], Masato Hibino[1], Ichi Takumi[1] and Tetsuo Kinoshita[2]

[1]Nagoya Institute of Technology
[2]Tohoku University
[1]t-uchiya@nitech.ac.jp hibino@uchiya.nitech.ac.jp takumi@nitech.ac.jp
[2]kino@riec.tohoku.ac.jp

Abstract

Agent-oriented computing is a technique to run a big-scale system by cooperation among plural agents. An agent with learning ability is called a "Learning Agent". A learning agent learns past actions, and changes the movement knowledge dynamically. A learning agent can solve a complicated problem in intellectual action. However, when a user wants to use a learning agent, little support exists for the development of a learning agent in an agent framework. Previous research proposed agent environment for developing single learning agent. However, effective multiagent learning environment has not realized yet, so agent developer has to implement learning function. To decrease the burden of developing the learning agents, in this paper, we newly introduce agent environment for developing cooperative learning multiagent into the DASH agent framework by applying the Nash-Q learning model. Moreover, we show the effectiveness of proposed mechanism through some experiments.

Keywords: *Agent Environment, Learning agent, Nash-Q Learning*

1. Introduction

In recent years, various needs have arisen along with the rapid spread of the internet. Requests have propagated to satisfy them. Systems change frequently. Therefore, the need exists for a flexible system performing in a processing fitting environment. Agent-oriented computing can resolve this problem. Actually, agent-oriented computing is a technology using autonomous agents. Agents with intelligence are called learning agents. Learning agents can behave with intelligent action. An agent framework is necessary for development of an agent system. However, few frameworks provide development support of learning agents. Developers must perform all the implementation to use learning agents. Therefore, they must learn technical knowledge related to machine learning, and it is necessary to encode it in all agents. These are severe burdens.

Previous research [1] proposed agent environment for developing single learning agent. However, effective multiagent learning environment has not realized yet, so agent developer has to implement learning function for cooperation of multiple agents. To decrease the burden of developing multiple learning agents, we newly propose agent environment for developing cooperative learning multiagent by applying the Nash-Q learning model.

This paper includes following chapters. Chapter 2 explains the DASH agent framework [2] that is basic agent environment of this research. Chapter 3 describe the previous research and shows the existing problems. Chapter 4 explains the Nash-Q learning model that is effective method in multiagent environment and explains proposal mechanism. Chapter 5 shows the experiment and evaluation. Chapter 6 depicts comparison of related works. Finally, Chapter 7 describe our research summary and future work.

2. DASH Agent Framework

In this research, the "Distributed Agent System based on Hybrid architecture (DASH) framework", an agent framework, is used. The DASH framework is a framework for a multi-agent system with a repository for agent storage and a workplace in which agents are running compose a platform to develop an agent system.

2.1. Outline

The DASH framework [2] is a framework for a multi-agent system with a repository for agent storage and a workplace in which agents are running compose a platform to develop an agent system Figure 1.

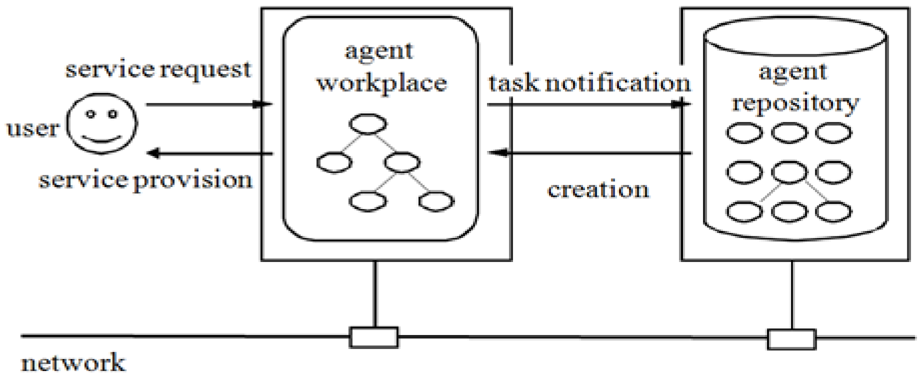


Figure 1. DASH Agent Framework

• **Agent Repository:** An agent repository stores agent-developer-produced agents that can act immediately according to the user's request. Additionally, it is a mechanism to generate an agent organization depending on the user's request. The concrete functions of the agent repository include acceptance of user requests, selection of agents, decisions related to agent organization, and generation of agent organizations operating on the agent workplace. The agent repository fully supports the agent system lifecycle through these processes.

• **Agent Workplace:** An agent workplace is an environment in which agent-repository-generated agent systems operate. An agent workplace offers a function by which an agent system exchanges messages with other agent groups and carries out cooperative problem solving based on the behavioral knowledge stored in each agent.

2.2. DASH Agent Architecture

An agent architecture of the DASH framework is presented in Figure 2.

An explanation about each mechanism is presented below.

Cooperation Mechanism (CM)

The CM exchange messages with another agent. It receives the message sent by other agents and it sends them to DK. The message format used in this architecture is based on the Knowledge Query Manipulating Language (KQML).

Domain Knowledge-base (DK)

DASH agent is a rule-based model agent. DK is a mechanism to decide an action Figure 3. Working Memory is storage of the agent's own parameter and environmental information. The DASH agent behavior is explained below.

1) The inference engine controls the operation-knowledge searches for a rule that matches the content that is reformed to WM.

2) The matched rule is executed.

3) It returns to procedure 1 when the content of WM is updated by executing the selected rule.

Task Processing Mechanism (TPM)

TPM is an interface module for controlling BP directly. Computational Base Process (BP) BP is a computational base process in agent. For instance, BP corresponds to a process of UNIX OS, a Java object, *etc*. The agent can use information provided from BP.

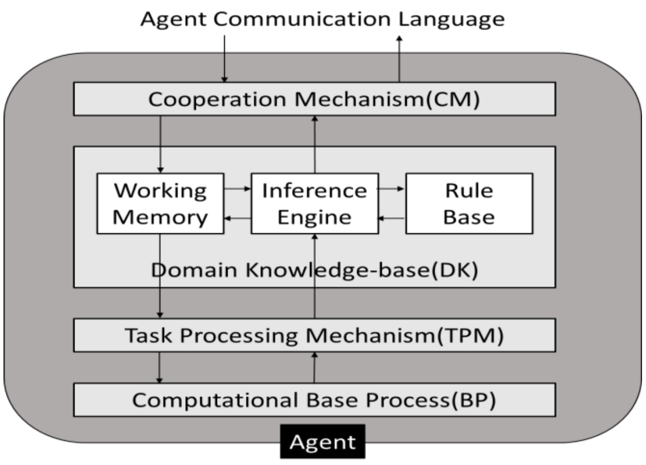

Agent Communication Language
Cooperation Mechanism(CM)
Working Memory
Inference Engine
Rule Base
Domain Knowledge-base(DK)
Task Processing Mechanism(TPM)
Computational Base Process(BP)
Agent

Figure 2. DASH Agent Architecture

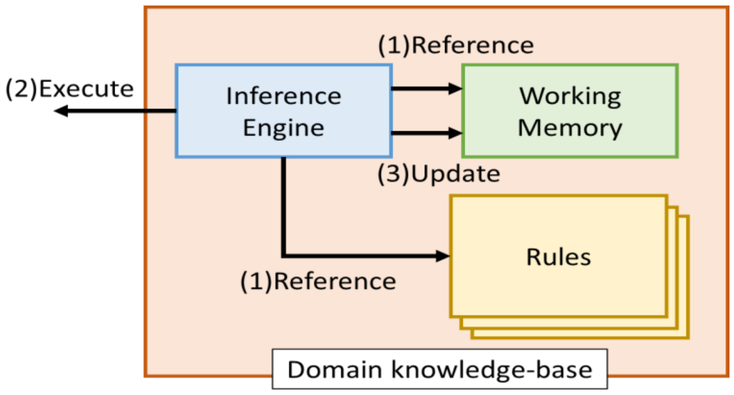


Figure 3. Domain Knowledge-base of the DASH Agent

3. Previous Research

This section provides an explanation of previous research [1][3].

3.1. Outline

Itazuro [1] realized the support of development of learning single agent by adding development support functions for learning agent to a DASH framework.

(F1) Automatic learning function

The rule priority is updated automatically using Q-Learning or Profit Sharing. Behavior of the automatic learning function is shown below Figure 4.

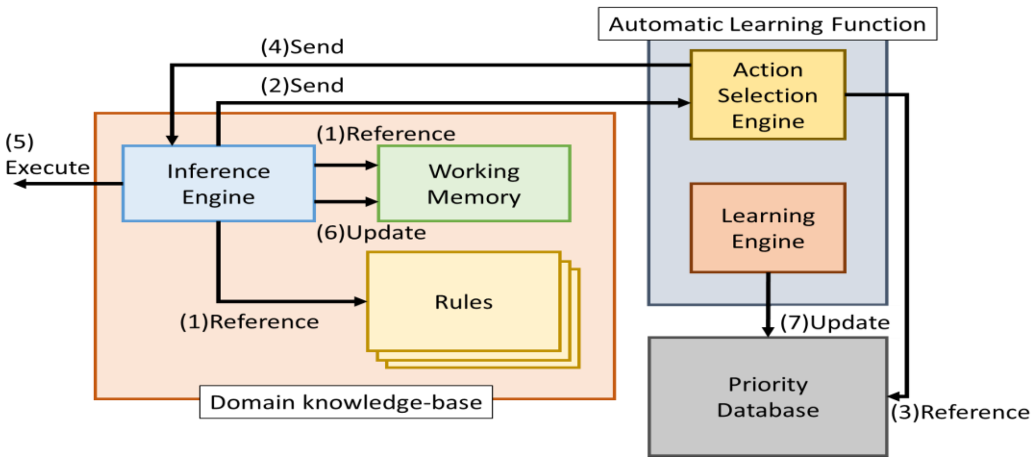


Figure 4. Automatic Learning Function of Previous Research

(1) Reference of the content of WM

The rule matching a current state is searched.

(2) Transmission of the available rule set

All matched rules are sent to the Action Selection Engine.

(3) Selection of the rule

The action is chosen.

(4) Send DK selected rule

The rule selected in (3) is sent to the inference engine.

(5) Execution of the rule

The rule sent in (4) is executed.

(6) Update of WM

The effect of the action is used.

(7) Update of the priority of the executed rule

The reward of the action is used for updating, and for judgment of the end.

(F2) Preservation and reference of learning data function

When an agent moves again, reopening of learning is enabled by reading the learning data file stored beforehand.

(F3) Automatic drawing of the graph function

This function drew a graph automatically to show the agent movement.

3.2 Agent Development Support

Previous research [3] proposed agent development environment called IDEAL Figure 5. IDEAL is the expansion of IDEA [4] that is developed by our research group to create the DASH agent efficiently. In order to support the design and implementation of DASH agents, IDEA has following four mechanisms.

・Mechanism of agent search support

・Mechanism of agent programming support

・Mechanism of agent simulation support

・Mechanism of agent registration support

Moreover, “Learning mechanism” is introduced and integrated to this environment. Consequently, the development work to create learning agent is more accelerated.

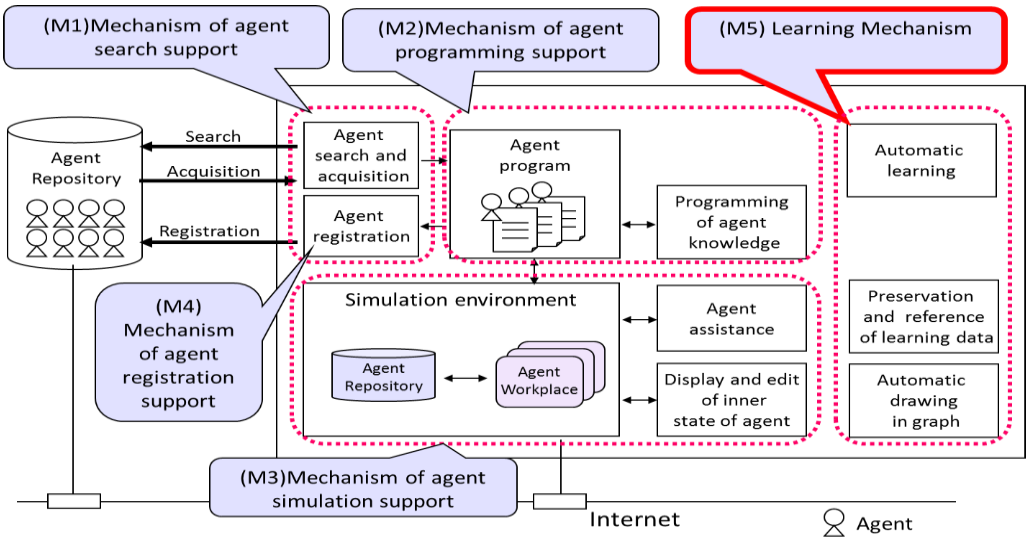

(M1)Mechanism of agent search support
(M2)Mechanism of agent programming support
(M5) Learning Mechanism
Search
Acquisition
Registration
Agent Repository
Agent search and acquisition
Agent registration
Agent program
Programming of agent knowledge
Automatic learning
(M4) Mechanism of agent registration support
Simulation environment
Agent Repository
Agent Workplace
Agent assistance
Display and edit of inner state of agent
Preservation and reference of learning data
Automatic drawing in graph
(M3)Mechanism of agent simulation support
Internet
Agent

Figure 5. IDEAL: Interactive Design Environment of Agent System with Learning Mechanism

3.3. Problems of Previous Research

(P1) There is unsuitable learning for the multi-agent system.

Q-Learning and Profit Sharing might impede other agents under multi-agents because they are unable to consider the actions of other agents.

(P2) Comparison of plural graphs is difficult.

A graph is produced by each agent, so the comparison of the graph in plural agents is difficult.

As a result, the support of learning agents is insufficient.

4. Proposal Mechanism

We solve existing problems by the expansion of previous research.

4.1. Introduction of Nash-Q Learning

As a solution for (P1), we suggest development support of a Nash-Q Learning agent. Q-Learning and Nash equilibrium compose Nash-Q Learning [5]. Nash-Q Learning predicts the strategies of other agents from the history of past action, and raises the priority of an action becoming a Nash equilibrium. The flow of the update priority is shown below. This algorithm is in [6]. One agent is agent A, other agent is agent B, “a” is action, “α” is learning rate, “γ” is reduction rate.

1) Agent A has $I^A(a^B|S)$ and $\bar{Q}^A(S,a^A)$. $I^A(a^B|S)$ is agent B’s strategy predicted by agent A. $\bar{Q}^A(S,a^A)$ is shown below.

$$\bar{Q}^A(S,a^A) \equiv \sum_{a^B \in A^B} I^A(a^B|S)Q^A(S,a^A,a^B)$$

In a current state ($s_t \in S$), agent A chooses an action according to a policy(ε-greedy, *etc.*).

2) Agent A will execute the action selected in procedure 1. Here, other agents act at the same time. By the action of all agents, the state switches to the next state(s_{t+1}), and agents receive reward (r_{t+1}^A) from the environment.

3) Agent A updates $Q(s_t, a_t^A, a_t^B)$ by following expression

$$Q(s_t, a_t^A, a_t^B) \leftarrow (1-\alpha)Q(s_t, a_t^A, a_t^B) + \alpha[r_{t+1}^A + \gamma \max_{a \in \mathcal{A}^A} \bar{Q}(s_{t+1}, a_{t+1}^A)]$$

And, updates I^A about $a^B \in \mathcal{A}^B$ by following expression.

$$I^A(a^B, s_t) \leftarrow (1-\theta)I(a^B|s_t) + \begin{cases} \theta & (a^B = a_t^B) \\ 0 & (otherwise) \end{cases}$$

θ is a parameter to ascertain how to use the observed action selected by agent B.

4) to t is added 1; then return to procedure 1 if it is not finished.

1) Working example:

We show the example that Nash-Q Learning is effective. Two agents exist in 3 × 3 grids. Each agent aims at the diagonal goal (Fig. 6). Details of this experiment are presented below.

• One trial ends if either agent arrives at its own goal.

• Even if two agents collide, they are reopened using an immediate state. One trial is not ended.

• Each agent receives a positive reward (+100) if they arrive at a goal. They get a negative reward (-10) if they collide.

• Agents have rules, “up”, “down”, “left”, “right”, and “stay”. They choose a rule, which is not to hit the wall.

In multi-agent systems, it is not desirable to interfere with other agents. This set up of rewards is therefore useful for multi-agent systems. We test it for two Q-Learning agents and two Nash-Q Learning agents. One trial is conducted at 1,000 times.

2) Result:

Table 1 shows the result. It is the average of the number of times which performed three trials. It is often case that an agent of either arrives at a goal as for the Q-Learning, and it is rare to arrive at a goal at the same time. Nash-Q Learning gets much number of times of Agent A and B each arrive at a goal simultaneously.

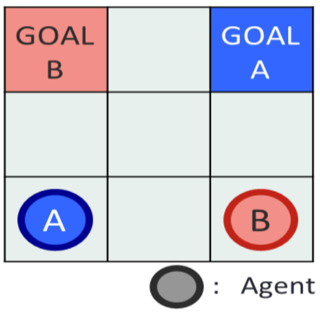


Figure 6. Route Search by Two Agents

Table 1. The Number of Times that Agents Arrive at Goal

	Nash-Q Learning agents	Q-learning agents
Agent A gets a goal	371	576.67
Agent B gets a goal	417	413
Agent A and B each get a goal simultaneously	212	10.33

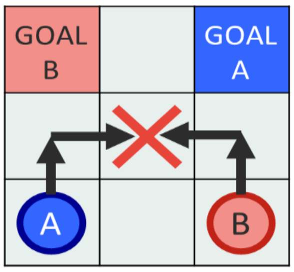


Figure 7. Movement of the Q-Learning Agent

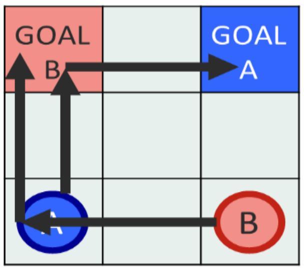


Figure 8. Movement of the Nash-Q Learning Agent

3) Consideration:

There is much number of times that Nash-Q Learning makes a goal simultaneously than Q-Learning. Because the Q-Learning agents consider only the profit of own, they cannot avoid other agents (Fig. 7). Nash-Q Learning agents accumulate the action history of other agents, so they can make mutual concessions Figure 8. In a cooperation action, Nash-Q Learning is superior to Q-Learning.

4.2. Knowledge Mechanism for Nash-Q Learning Agent

Development support of Nash-Q Learning is introduced by expansion to an automatic learning function of earlier research. This supports the development of a Nash-Q Learning agent, and reduces the burden on the developer. Nash-Q Learning can learn an action and avoid disturbing other agents. Figure 9 depicts a concept diagram of an expanded automatic learning function. We describe the flow of development support for Nash-Q Learning.

(1) Reference of the content of WM

The rule matching a current state is sought.

(2) Transmission of the available rule set

All matched rules are sent to the Action Selection Engine.

(3) Selection of the rule

The action is chosen.

(4) Send DK selected rule

The rule selected in (3) is sent to the inference engine.

(5) Execution of the rule
The rule sent in (4) is executed.
(6) Update of WM
The effect of the action is used.
(7) Update of the other agent database
The action selected by the other agent is used.
(8) Reference of other agent database
When a priority is updated, the learning engine uses information of the other agent's database.
(9) Update of the priority of the executed rule
The reward of the action is used for updating, and for judgment of the end.

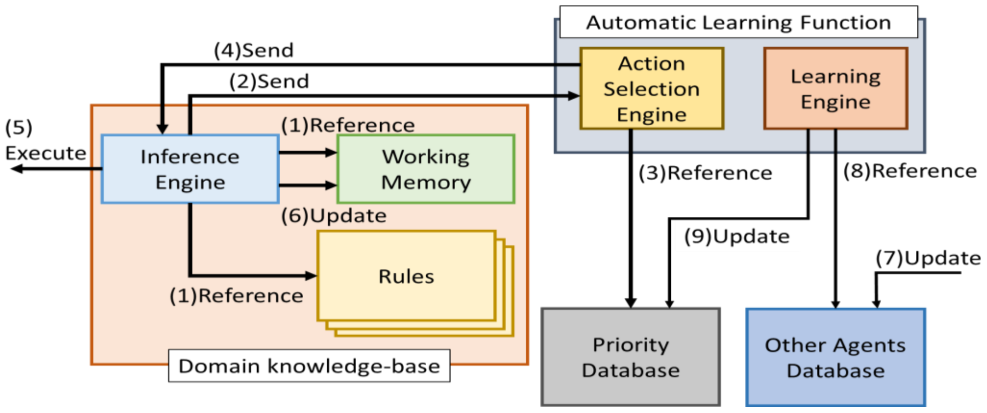


Figure 9. Automatic Learning Function of the Proposed Mechanism

We explain the mechanism we added.

1) Other agent database:

It is used only when development support of the Nash-Q Learning agent is used. Whenever other agents act, the other agents' databases store a history of the action. The learning engine refers to it for update priority.

2) Learning engine:

The learning engine examined in previous studies is expanded to use Nash-Q Learning. After its own actions, it updates the priority of the rule it chose. A developer can select Q-Learning, Profit Sharing, and Nash-Q Learning for each system.

4.3. Improvement of Automatic drawing of the graph function

Collection of the graphs made by agents is a solution for (P2). Furthermore, a graph viewer that can plot the learning data chosen by the user is added. The developer can compare the result with the past learning data. Details of each function are shown below.

1) Collection graph:

Moving of automatic drawing of the graph function to the framework from each agent produces a drawing of the information that all agents in the environment have. In this way, graphs are displayed on two axes. Thereby, a comparison graph can be produced easily Figure 10.

2) Graph viewer:

The developer can compare a learning agent and learning agents which worked in the past on this function. In the previous research, Gnuplot was used for graph drawing, but we introduced new graph viewer in the framework to get rid of installation.

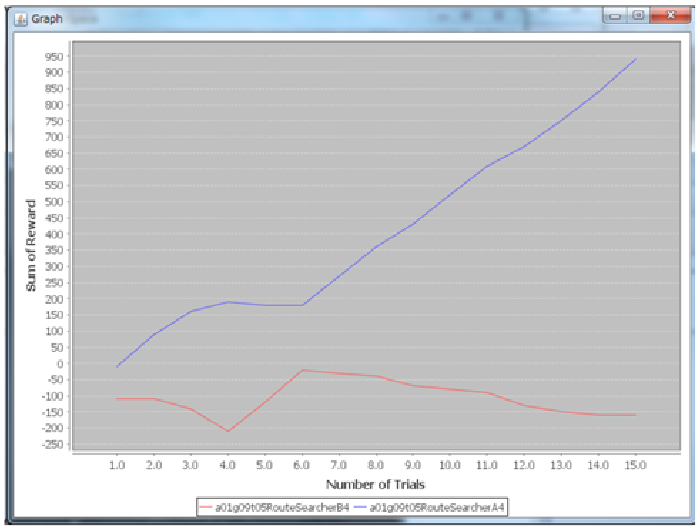


Figure 10. Graph for Grasping the Total Rewards

4.4. Addition of Automatic Parameter Adjustment Function

We added a function to coordinate three parameters (α,γ,θ) automatically. When developers use Nash-Q Learning, they can use Nash-Q Learning agents without knowledge of machine learning.

5. Experiment and Evaluation

We performed two evaluation experiments for the proposed mechanism.

5.1. Evaluation Experiment 1

We performed an evaluation to ascertain the quantity of code to describe one Nash-Q Learning agent.

1) Outline:

We calculated the rate of reduction of quantity of description. Here, A is the quantity of description necessary to use the agents with the conventional DASH framework. Also, B is the quantity of description necessary to make Nash-Q learning agents with the proposed mechanism. It is noteworthy that A and B do not include the rule description. They are quantities of description necessary to introduce learning characteristics into an agent. The rate of reduction is independent of the environment. The unit is the steps.

2) Result:

Results are shown in Table 2. Rate of reduction is 98.5%. Then, we confirmed the reduction of developer burden.

Table 2. Reduction Rate for Agent Programming

Amount A (steps)	Amount B (steps)	Reduction rate (%)
392	6	98.5%

5.2. Evaluation Experiment 2

We performed a questionnaire to ascertain whether proposal mechanism is useful. We evaluate the easiness of creating Q-Learning agent and Nash-Q learning agent.

1) Outline:

We asked ten students of our research group use it to evaluate it in 1-5.

2) Result: Results are presented in Table 3. All subjects of the questionnaire responded that proposal mechanism execute great support to create learning agents. We confirmed high usefulness of our proposal mechanism.

Table 3. Evaluation of Usefulness

	Average	Standard deviation
Ease of creating a Nash-Q Learning agent	4.3	0.9
Ease of creating a Q-learning agent	4.3	0.9

5.2. Evaluation Experiment 3

We performed a questionnaire to ascertain whether an improvement of “Automatic draw of the graph function” and “Automatic parameter adjustment function” are useful.

1) Outline:

We asked six students of our research group use it to evaluate it in 1-5.
2) Result: Results are presented in Table 4. All subjects of the questionnaire responded that this improvement is appropriate. We confirmed high usefulness for this function.

Table 4. Evaluation of New Functions

	Average	Standard deviation
Ease of using graph viewer	4.0	0.58
Usefulness of automatic parameter Adjustment function	4.67	0.48

6. Comparison of Related Work

We compared our proposal mechanism with famous agent frameworks. Comparison target are IDEA[4], IDEAL[3], ABLE[7], JADE[8], OMAS[9].

Evaluation points are three items, the one is development support of single agent learning. Next one is development support of multiagent learning. Last one is availability of software.

We show the comparison result in Figure 11.

IDEA is agent design environment for multiagent system developed by our research group. There are no support functions to develop learning agents.

IDEAL is focus on the support of single agent learning, so support of multiagent learning are few.

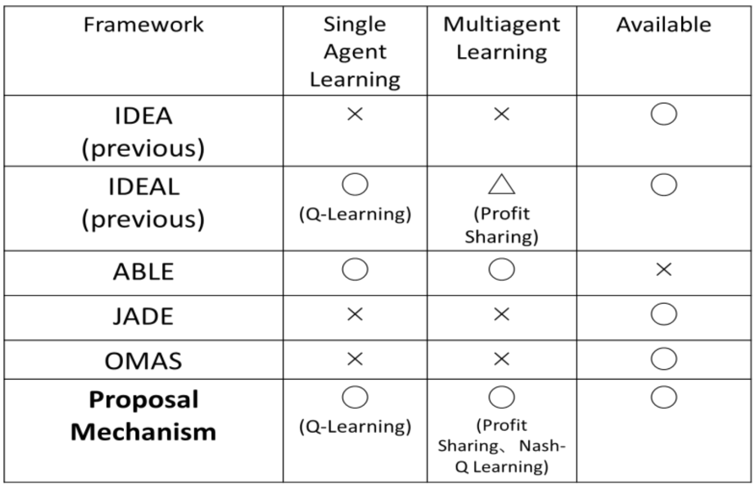
ABLE produced by IBM supports both single agent learning and multiagent learning, but software provisioning has finished and software is not available now.

JADE produced by Telecom Italia Lab is famous agent framework in Europe and FIPA-compliant standard agent platform. This software is available but there are no support functions to develop learning agents.

OMAS produced by France research group is lisp based agent framework. This software is available but there are no support functions to develop learning agents.

Our proposal mechanism is available now. Moreover, this mechanism support not only single agent learning, but also multiagent leaning.

As a result, proposal mechanism is best agent framework to creating intelligent multiagent system.



Framework	Single Agent Learning	Multiagent Learning	Available
IDEA (previous)	×	×	○
IDEAL (previous)	○ (Q-Learning)	△ (Profit Sharing)	○
ABLE	○	○	×
JADE	×	×	○
OMAS	×	×	○
Proposal Mechanism	○ (Q-Learning)	○ (Profit Sharing、 Nash-Q Learning)	○

Figure 11. Comparison among Agent Frameworks

7. Conclusion

This research was conducted to realize sufficient development support of a learning agent. We realized the proposed mechanism by expansion of results of previous research. We also confirmed the usefulness of each function through evaluation experiments. Future work are refinement of GUI, shorten of learning time and application to creation of learning robots.

References

[1] S. Itazuro, T. Uchiya, I. Takumi and T. Kinoshita, “Design Environment of Reinforcement Learning Agents for Intelligent Multiagent System”, Proceedings of BWCCA2012, **(2012)**, pp. 668-672.

[2] K. Sugawara, H. Hara, T. Kinoshita and T. Uchiya, “Flexible Distributed Agent System programmed by a Rule-based Language”, Proceedings of the Sixth IASTED International Conference of Artificial and Soft Computing, **(2002)**, pp. 7-12.

[3] T. Uchiya, S. Itazuro, I. Takumi and T. Kinoshita, “IDEAL: Interactive Design Environment for Agent System with Learning Mechanism”, Proceeding of the 12th IEEE International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC2013), **(2013)** July, pp. 153-160.

[4] T. Uchiya, H. Hara, K. Sugawara and T. Kinoshita, “Repository-Based Multiagent Framework for Developing Agent Systems”, Transdisciplinary Advancements in Cognitive Mechanisms and Human Information Processing, Ch.4, IGI Global, ISBN 9781609605537, EISBN13: 9781609605544, **(2011)**, pp. 60-79.

[5] J. Hu and M. P. Wellman, “Nash Q-Learning for General-Sum Stochastic Games”, Journal of Machine Learning Research, vol. 4, **(2003)**, pp. 1039-1069.

[6] S. Kitahara, Y. Tanigawa and H. Tsuruoka, “Emergence of Cooperative Action in Nash-Q Learning”, Res. Bull Fukuoka Inst., vol. 40, no. 1, **(2007)**.

[7] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III and Y. Diao, “ABLE: A toolkit for building multiagent autonomic systems”, in IBM Systems Journal, vol. 41, no. 3, **(2002)**, pp. 350-371.

[8] JADE - Java Agent Development Framework, http://jade.tilab.com/.

[9] OMAS platform, http://www.utc.fr/~barthes/OMAS/.

Authors

Takahiro Uchiya, is an Associate Professor of the Information Technology Center, Nagoya Institute of Technology, Nagoya, Japan. He received a Ph.D. degree from Tohoku University in 2004. His research interests include knowledge engineering and design methodologies of agent system. Dr. Uchiya is a member of IEEE, IPSJ and IEICE.

Masato Hibino, is a Master course student of the Department of Computer Science and Engineering, Graduate School of Engineering, Nagoya Institute of Technology, Nagoya, Japan. His research interests include agent-oriented software computing.

Ichi Takumi, is a Professor of the Information Technology Center, Nagoya Institute of Technology, Nagoya, Japan. His research interests include knowledge computer system networks, fundamental informatics, intelligent informatics, measurement engineering, communication/ network engineering, and natural disaster science.

Tetsuo Kinoshita, is a Professor at the Research Institute of Electrical Communication, Tohoku University, Japan. He received a Dr. Eng. degree in information engineering from Tohoku University in 1993. His research interests include agent engineering, knowledge engineering, knowledge-based systems and agent-based systems. He received the IPSJ Research Award, the IPSJ Best Paper Award, and the IEICE Achievement Award respectively in 1989, 1997, and 2001. Dr. Kinoshita is a member of IEEE, ACM, AAAI, IEICE, IPSJ, and JSAI.