Bio-inspired Self-Adaptive Agents in Distributed Systems

Ichiro Satoh
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

KEYWORD

Adaptive Agents
Distributed Systems

ABSTRACT

This paper proposes a bio-inspired middleware for selfadaptive software agents on distributed systems. It is unique to other existing approaches for software adaptation because it introduces the notions of differentiation, dedifferentiation, and cellular division in cellular slime molds, e.g., dictyostelium discoideum, into real distributed systems. When an agent delegates a function to another agent coordinating with it, if the former has the function, this function becomes less-developed and the latter's function becomes welldeveloped.

1 Introduction

Cellular differentiation is the mechanism by which cells in a multicellular organism become specialized to perform specific functions in a variety of tissues and organs. Different kinds of cell behaviors can be observed during embryogenesis: cells double, change in shape, and attach at and migrate to various sites within the embryo without any obvious signs of differentiation. The mechanism is essential in the development of a complex organism.

This paper aims at introducing the notion of differentiation into a distributed system as a mechanism for adapting software components, which may be running on different computers connected through a network. Software components in existing distributed systems only continue to offer their initial functions. Therefore, when constructing a distributed application, we need to initially define the role of each of its components. However, it is almost impossible to exactly know the functions that each of the components should provide, since distributed systems are dynamic and may partially have malfunctioned, e.g., network partitioning.

In the remainder of this paper, we discuss the requirements of the framework through reviewing related work (Section 2), the design of our framework (Section 3), and an implementation of the framework (Section 4). We explain our evaluation of the framework with some applications (Section 5) and provide a summary, discuss some future issues (Section 6).

2 Related work

Several researchers have explored evolutional computing approaches, including genetic computation and genetic programming [9] and swarm intelligence [3, 5]. Many mechanisms from which self-organization emerges are often too diverse, when the are applied to real distributed systems whose structures and applications may be dynamically changing. However, real systems may have no chance of ascertaining the fitness of randomly generated parameters or programs, because they have an effect on the real world and are used for mission-critical processing. Since the size and structure of real distributed systems have been designed and optimized to the needs of their applications, the systems have no room to execute such large numbers of swarm agents. Consequently, our software adaptation mechanism for distributed systems must involve as few computational resources as possible that the systems spend for software adaptation.

There have been several attempts to support software adaptation in the literatures on self-organizing properties, autonomic computing, and software engineering. Autonomic computing was initiated by IBM and has encouraged research on providing self-organizing properties to systems. Several existing studies primarily support middleware or higher layers as models and system architecture in a distributed computing setting like ours. Bigus et al. [1] proposed an agent-based toolkit for autonomic systems, where each agent has a closedloop controller as part of the whole hierarchy of distributed control. The toolkit was intended to customize groups of agents but not the functions inside agents. Blair et al. [2] tried to introduce self-awareness and self-healing into a CORBAcompatible Object Request Broker (ORB). Their system was a meta-level architecture with the ability of dynamically binding CORBA objects. Jaeger et al. [8] introduced the notion of self-organization to ORB and a publish/subscribe system.

Georgiadis et al. [6] presented connection-based architecture for self-organizing software components on a distributed system. Like other software component architectures, they intended to customize their systems by changing connections between components instead of internal behaviors inside components. Like ours, Cheng at al. [4] presented an adaptive selection mechanism for servers by enabling selection policies, but they did not customize the servers themselves. They also needed to execute different servers simultaneously.

Suda et al. proposed bio-inspired middleware, called BioNetworking, for disseminating network services in dynamic and large-scale networks where there were a large number of decentralized data and services [10, 15]. Although they introduced the notion of energy into distributed systems and enabled agents to be replicated, moved, and deleted according to the number of service requests, they had no mechanism to adapt agents' behavior unlike ours. Most of their parameters, e.g., energy, tend to depend on a particular distributed system. so that they may not be available in another system[1]. Our framework should be independent of the capabilities of distributed systems as much as possible.

We proposed a nature-inspired approach to dynamically deploying agents at computers in our previous papers [11, 12]. The approach enabled each agent to describe its own deployment as a relationship between its location and another agent's location. However, the approach had no mechanism for differentiating or adapting agents themselves.

3 Background

The basic inspiration for our approach lies in the development of multicellular organisms in nature. Cellular differentiation is the process by which a less specialized cell develops or matures to possess a more distinct form and function in developmental biology. For example, cellular slime molds, e.g., dictyostelium discoideum, are eukaryotic microorganisms in the soil. They are solitary amoebae and feed on bacteria. Once food becomes scarce, the cells start to aggregate and differentiate themselves. Each amoebae secretes cyclic-adenosinemonophosphate (cAMP) that chemotaxically leads the cells to aggregate. This multicellular mound goes on to form a colony. The mound has two distinct cell types: prespore cells and prestalk cells. Each cell initially intends to become the former and periodically secretes cAMP that chemotaxically leads other cells to the latter. Secreted cAMP only affects cells within a certain distance (6 to 10 cell diameters from the signaling cell). As a result, prespore cells comprise about 80% of the mound, while prestalk cells account for the remaining 20% of the mound. The mound becomes a mass of aggregated amoebae with the ability to move to a more desirable place. Differentiation into mature spore and stalk cells proceeds. While stalk cells are programmed to die, spore cells germinate into single-celled amoebae. Dedifferentiation is the process by which a partially or terminally differentiated cell reverts to an earlier developmental stage. The process is often seen in more basal life forms such as worms and amphibians. It may occur before the regeneration of appendages in plants and certain animals.

[1] To prevent malicious agents from being passed between computers, each runtime system supports a Kerberos-based authentication mechanism for agent migration. Since it can inherit the security mechanisms provided in the Java language environment, the Java VM explicitly restricts agents so that they can only access specified resources to protect computers from agents.

4 Approach

The framework assumes that a distributed application running on different computers consist of (autonomous) programmable entities, called agents, correspondings to cells. It emulates the above process between agents, which may be running on different computers. It also introduces the undertaking/delegation of functions in agents from/to other agents as their differentiation factor. Agents are differentiated according to demands from other agents. When an agent receives a request message to do a function from another agent, the function is more developed in the former and it is less developed in the latter (Figure 1). Finally, agents are only specialized to some of their functions. They undertake the specialized functions from other agents. Instead, they delegate other functions, which may be initially provided in them, to other agents that can provide the functions. Each agent has one or more functions with weights, where each weight corresponds to the amount of cAMP and indicates the superiority of its function.

Agents may lose their functions due to differentiation as well as be busy or failed. The approach also offers a mechanism to recover from such problems based on dedifferentiation, which a mechanism for regressing specialized cells to simpler, more embryonic, unspecialized forms. As in the dedifferentiation process, if there are no other agents that are sending restraining messages to an agent, the agent can perform its dedifferentiation process and strengthen their lessdeveloped or inactive functions again.

5 Design and Implementation

Our approach is maintained through two parts: runtime systems and agents. The former is a middleware system for running on computers and the latter is a self-contained and autonomous software entity. It has three protocols for (de)differentiation and delegation.

5.1 Agent

Each agent consists of one or more functions, called the behavior parts, and its state, called the body part, with information for (de)differentiation, called the attribute part.

- The body part maintains program variables shared by its behaviors parts like instance variables in object orientation. When it receives a request message from an external system or other agents, it dispatches the message to the behavior part that can handle the message.
- The behavior part defines more than one application specific behavior. It corresponds to a method in object orientation. As in behavior invocation, when a message is received from the body part, the behavior is executed and returns the result is returned via the body part.
- The attribute part maintains descriptive information with regard to the agent, including its own identifier. The attributes contains a database for maintaining the weights of its own behaviors and for recording information on the behaviors that other agents can provide.

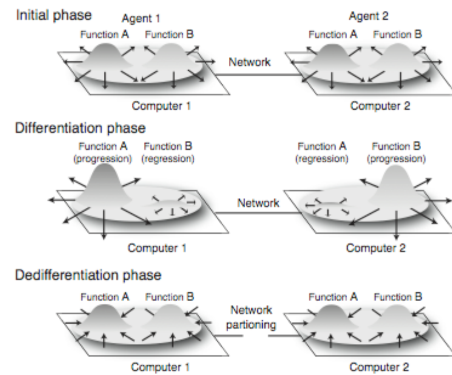

Figure 1: Cellular Differentiation-like approach for Software Adaptation

There is no universal selection function, ϕ , for mapping from the weights of behaviors to at most one appropriate behavior like that in a variety of creatures. Instead, the approach permits agents to use their own evaluation functions, because the selection of behaviors often depends on their applications. For example, one of the simplest evaluation functions makes the agent that wants to execute a behavior select one whose weight has the highest value and whose signature matches the wanted behavior if its database recognizes one or more agents that provide the same behavior, including itself.

The agent has behaviors $b^k_1,\ldots,b^k_n$ and wik is the weight of behavior b^k_i. Each agent (k-th) assigns its own maximum to the total of the weights of all its

behaviors. The W^k_i is the maximum of the weight of behavior b^k_i. The maximum total of the weights of its behaviors in the k-th agent must be less than W^k. ($W^k \geq \Sigma^n_{i=1} w^k_i$) , where $w^k_j - 1$ is 0 if w^k_j is 0. The W^k may depend on agents. In fact, W^k corresponds to the upper limit of the ability of each agent and may depend on the performance of the underlying system, including the processor. Note that we never expect that the latter will be complete, since agents periodically exchange their information with neighboring agents. Furthermore, when agents receive no retraining messages from others for longer than a certain duration, they remove information about them.

5.2 Function invocation

When an agent wants to execute a behavior, it needs to select one of the available behaviors (b^j_i ,...,b^m_i), even if it has the behavior, according to the values of their weights. This involves three steps.

- When an agent (*k-th* agent) wants to execute behavior bi, it looks up the weight (w^i_k) of the same or a compatible behavior from its database and the weights (w^i_j,...,w^i_m) of such behaviors (b^j_i,...,b^m_i) from the database.
- If multiple agents, including itself, can provide the wanted behavior, the k-th agent selects one of the agents according to selection function ϕ^k , which maps from w^i_k and w^i_j,...,w^i_m to b^l_i, where l is k or j,...,m.
- The *k-th* agent delegates the selected agent to execute the behavior b^l_i and waits for the result from the *l-th* agent.

There is no universal selection function, ϕ , for mapping from the weights of behaviors to at most one appropriate behavior like that in a variety of creatures. Instead, the approach permits agents to use their own evaluation functions, because the selection of behaviors often depends on their applications. For example, one of the simplest evaluation functions makes the agent that wants to execute a behavior select one whose weight has the highest value and whose signature matches the wanted behavior if its database recognizes one or more agents that provide the same behavior, including itself.

5.3 Differentiation

The approach introduces the undertaking/delegation of behaviors in agents from other agents as a differentiation factor. Behaviors in an agent, which are delegated from other agents more frequently, are well developed, whereas other behaviors, which are delegated from other agents less frequently, in the cell are less developed. Finally, the agent only provides the former behaviors and delegates the latter behaviors to other agents. Our differentiation mechanism consists of two phases. The first involves the progression of behaviors in three steps.

- When an agent (*k-th* agent) receives a request message from another agent, it selects the behavior (b^k_i) specified in the message from its behavior part and dispatches the message to the selected behavior. It executes the b^k_i behavior and returns the result.
- The *k-th* agent increases weight w^k_i of the b^k_i behavior.
- The k-th agent multicasts a restraining message with the signature of the behavior, its identifier (k), and the behavior's weight (w^k_i) to other agents.

When behaviors are internally invoked by their agents, their weights are not increased. If the total weights of the agent's behaviors, Σw^k_i , is equal to their maximal total weight W^k, it decreases one of the minimal (and positive) weights (w^k_j is replaced by $w^k_j - 1$ where $w^k_j = min(w^k_1,...,w^k_n)$ and $w^k_j \geq 0$). Although restraining messages correspond to the diffusion of cAMP in differentiation, they can explicitly carry the weights of the agents that send them to reduce the number of restraining messages, because they can be substituted for more than one retaining message without weights. The second phase supports the retrogression of behaviors in three steps.

When an agent (*k-th* agent) receives a restraining message with regard to b^j_j from another agent (*j-th*), it looks for the behaviors (b^k_m , ... b^k_l) that can have the signature specified in the received message.

If it has such behaviors, it decreases their weights (w^k_m,..., w^k_l) in its database and updates the weight (w^j_i) in its database.

If the weights (w^k_m,...,w^k_l) are under a specified value, e.g., 0, the behaviors (b^k_m,...,b^k_l) are inactivated.

5.4 Dedifferentiation

Distributed systems may be damaged or stop due to disasters and problems. We need a mechanism for detecting and remedying failures in networking, computers, agents, remote computers, and other agents. To do this, each agent (*j-th*) periodically multicasts messages, called heartbeat messages, for behavior (b^j_i), which is still activated with its identifier (*j-th*). This involves two cases.

i) When an agent (*k-th*) receives a heartbeat message with regard to behavior (b^j_i) from another agent (*j-th*), it retains the weight (w^j_i) of the behavior (b^j_i) in its second database.

ii) When an agent (*k-th*) does not receive any heartbeat messages with regard to behavior (b^j_i) from another agent (*j-th*) for a specified time, it automatically decreases the weight (w^j_i) of the behavior (b^j_i) in its second database, and resets the weight (w^k_i) of the behavior (b^k_i) to the initial value or increases the weight (w^k_i) in its first database.

Note that behavior b^k_i is provided by the *k-th* agent and behavior b^j_i is provided by the *j-th* agent. The weights of behaviors provided by other agents automatically decrease without any heartbeat messages from the agents. Therefore, when an agent terminates or fails, other agents decrease the weights of the behaviors provided by the agent. If they have the same or compatible behaviors, they can then activate the behaviors, which may be inactivated. After a request message is sent to another agent, if the agent waits for the result to arrive for longer than a specified time, it selects one of the agents that can handle the message from its database and requests the selected agent. If there are no agents that can provide the behavior that can handle the behavior quickly, it promotes other agents that have the behavior in less-developed form (and itself if it has the behavior).

5.5 Implementation

Each runtime system is constructed as a middleware system with Java based on several technologies of mobile agent platforms [13]. It is responsible for executing agents and exchanging messages in runtime systems on other computers through a network[2][3]. It allow each agent, i.e., component, to have at most one activity through the Java thread library. It cooperates with an existing web server, called Jetty, to receive requests from external systems and return the results of agents through HTTP-based protocols.

When a runtime system detects failure in another, it removes information, including weights, about the behaviors of agents running on the systems. When a runtime system is (re)connected to a network, it multicasts heartbeat messages to other runtime systems to advertise itself, including its network address.

Our mechanism itself must be tolerant to network or system problems, e.g., network partitioning and node failure. The approach classifies communications, which may be between different computers, into two types. The former supports system-level communications between runtime systems, i.e., agent migration, and application-specific communications, i.e., request and reply messages. It is implemented through TCP sessions as reliable communications. When typical network problems occur, e.g., network partitioning and node failure during communication, the TCP session itself can detect such problems and it notifies runtime systems on the both sides to execute the exception handling defined in runtime systems or agents.

The latter supports messages for differentiation-based adaptation, e.g., restraining and heartbeat messages. These messages are transmitted as multicast UDP packets, which are unreliable. Restraining messages for behaviors that do not arrive at agents do not seriously affect our differentiation, because such messages decrease weights regarding behaviors but do not increase the weights. Since our mechanism does not assume that each agent has complete information about all agents, it is available even when some heartbeat messages are lost.

---

2 Our system was implemented independently of any existing middleware or frameworks for component oriented computing. But, it enabled us to implement components as JavaBeans components.

3 To prevent malicious agents from being passed between computers, each runtime system supports a Kerberos-based authentication mechanism for agent migration. Since it can inherit the security mechanisms provided in the Java language environment, the Java VM explicitly restricts agents so that they can only access specified resources to protect computers from agents.

6 Evaluation

Although the current implementation was not constructed for performance, we evaluated that of several basic operations in a distributed system where eight computers (Intel Core 2 Duo 1.83 GHz with MacOS X 10.6 and J2SE version 6) were connected through a giga-ethernet. The cost of transmitting a heartbeat or restraining message through UDP multicasting was 11ms. The cost of transmitting a request message between two computers was 22 ms through TCP. These costs were estimated from the measurements of round-trip times between computers. We assumed in the following experiments that each agent issued heartbeat messages to other agents every 100 ms through UDP multicasting.

The first experiment was carried out to evaluate the basic ability of agents to differentiate themselves through interactions in a reliable network. Each agent had three behaviors, called A, B, and C. The A behavior periodically issued messages to invoke its B and C behaviors or those of other agents every 200ms and the B and C behaviors were null behaviors. Each agent that wanted to execute a behavior, i.e., B or C, selected a behavior whose weight had the highest value if its database recognized one or more agents that provided the same or compatible behavior, including itself. When it invokes behavior B or C and the weights of its and others behaviors were the same, it randomly selected one of the behaviors. We assumed in this experiment that the weights of the B and C behaviors of each agent would initially be five and the maximum of the weight of each behavior and the total maximum W^k of weights would be ten.

Figure 2 presents the results we obtained from the experiment. Both diagrams have a timeline in minutes on the x-axis and the weights of behavior B in each agent on the y-axis. Differentiation started after 200ms, because each agent knows the presence of other agents by receiving heartbeat messages from them. Figure 2 (a) details the results obtained from our differentiation between two agents. Their weights were not initially varied and then they forked into progression and regression sides. Figure 2 (b) shows the detailed results of our differentiation between four agents and Figure 2 (c) shows those of that between eight agents. The results in (b) and (c) fluctuated more and then converged faster than those in (a), because the weights of behaviors in four are increased or decreased more than those in two agents. Although the time of differentiation depended on the period of invoking behaviors, it was independent of the number of agents. This is important to prove that this approach is scalable.

Our parameters for (de)differentiation were basically independent of the performance and capabilities of the underlying systems. For example, the weights of behaviors are used for relatively specifying the progression/repression of these behaviors.

The second experiment was carried out to evaluate the ability of the agents to adapt to two types of failures in a distributed system (3). The first corresponded to the termination of an agent and the second to the partition of a network. We assumed in the following experiment that three differentiated agents would be running on different computers and each agent had four behaviors, called A, B, C, and D, where the A behavior invokes other behaviors every 200 ms. The maximum of each behavior was ten and the agents' total maximum of weights was twenty. The initial weights of their behaviors (w^i_B, w^i_C, w^i_D) in *i-th* agent were (10, 0, 0) in the first, (0, 10, 0) in the second, and (0, 0, 10) in the third.

7 Conclusion

This paper proposed a framework for adapting software agents on distributed systems. It is unique to other existing software adaptations in introducing the notions of (de)differentiation and cellular division in cellular slime molds, e.g., dictyostelium discoideum, into software agents. When an agent delegates a function to another agent, if the former has the function, its function becomes less-developed and the latter's function becomes well-developed. When agents have many requests from other agents, they create their daughter agents. The framework was constructed as a middleware system on real distributed systems instead of any simulation-based systems. Agents can be composed from Java objects.

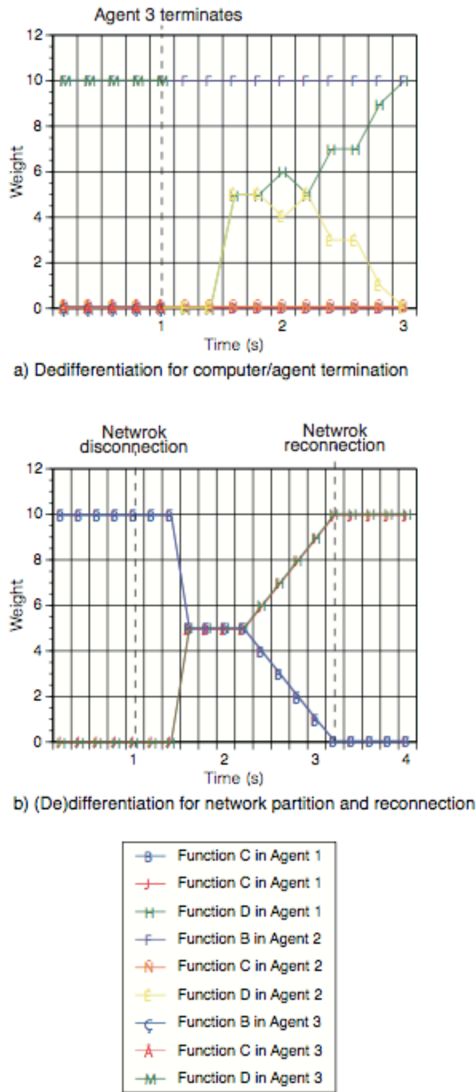

Figure 2: Degree of progress in differentiation-based adaptation

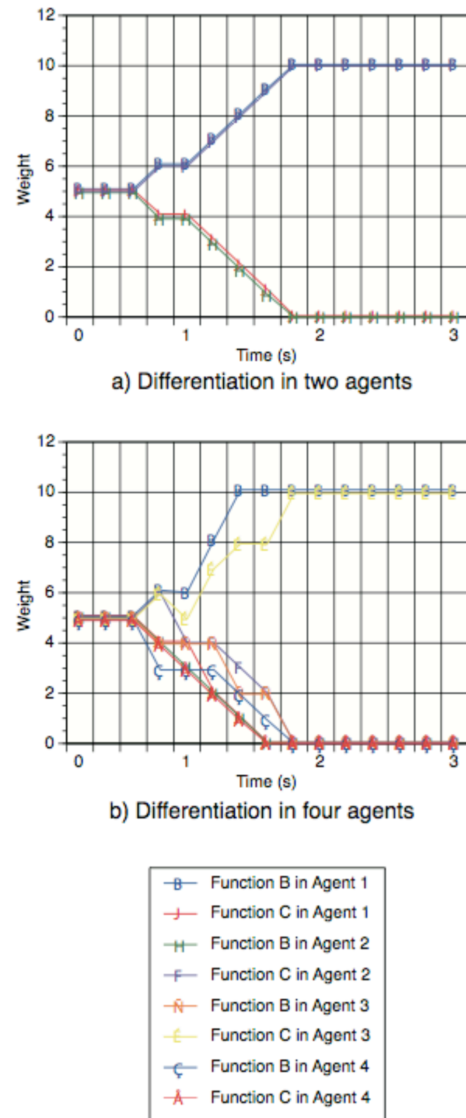

Figure 3: Degree of progress in adaptation to failed agent

9 References

[1] J.P. Bigus, D.A. Schlosnagle, J.R. Pilgrim, W.N. Mills, Y. Diao: ABLE: A toolkit for building multiagent autonomic systems, IBM Systems Journal vol.41, no.3, pp.350-371, IBM, 2002.

[2] G. S. Blair, G. Coulson, L. Blair, H. Duran-Limon, P. Grace, R. Moreira, and N. Parlavantzas: Reflection, selfawareness and self-healing in OpenORB, in Proceedings of 1st Workshop on Self-healing systems (WOSS'2002), pp.9-14, ACM Press, 2002.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz: Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, 1999.

[4] S. Cheng, D. Garlan, B. Schmerl: Architecture-based self-adaptation in the presence of multiple objectives, in Proceedings of International Workshop on Selfadaptation and Self-managing Systems (SEAMS'2006), pp.2-8, ACM Press, 2006.

[5] M.Dorigo and T. Stutzle: Ant Colony Optimization,MIT Press, 2004.

[6] I. Georgiadis, J. Magee, and J. Kramer: Self-Organising Software Architectures for Distributed Systems in Proceedings of 1st Workshop on Self-healing systems (WOSS'2002), pp.33-38, ACM Press, 2002.

[7] K. Herrman: Self-organizing Replica Placement A
Case Study on Emergence, in Proceedings of 2nd
IEEE International Conference on Self-Adaptive and 10 Self-Organizing Systems (SASO'2007), pp.13-22, IEEE
Computer Society, 2007.

[8] M. A. Jaeger, H. Parzyjegla, G. Muhl, K. Herrmann: Self-organizing broker topologies for publish/subscribe systems, in Proceedings of ACM symposium on Applied Computing (SAC'2007), pp.543-550, ACM, 2007.

[9] J.R. Koza: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992

[10] T. Nakano and T. Suda: Self-Organizing Network Services With Evolutionary Adaptation, IEEE Transactions on Neural Networks, vol.16, no.5, pp.1269-1278, 2005.

[11] I Satoh: Self-organizing Software Components in Distributed Systems, in Proceedings of 20th International Conference on Architecture of Computing Systems System Aspects in Pervasive and Organic Computing (ARCS'07), Lecture Notes in Computer Science (LNCS), vol.4415, pp.185-198, Springer, March 2007.

[12] I Satoh: Test-bed Platform for Bio-inspired Distributed Systems, in Proceesings of 3rd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems, November 2008.

[13] I Satoh: Mobile Agents, Handbook of Ambient Intelligence and Smart Environments, pp.771-791, Springer 2010.

[14] P. L. Snyder, R. Greenstadt, G. Valetto: Myconet: A Fungi-Inspired Model for Superpeer-Based Peer-to-Peer Overlay Topologies, in Proceedings of 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'2009), pp.40-50, 2009.

[15] T. Suda and J. Suzuki: A Middleware Platform for a Biologically-inspired Network Architecture Supporting Autonomous and Adaptive Applications. IEEE Journal on Selected Areas in Communications, vol.23, no.2, pp.249-260, 2005.