CELLULAR DIFFERENTIATION-BASED APPROACH FOR DISTRIBUTED SYSTEMS

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

Keywords: Disaggregated computing, Ubiquitous computing, Differentiation, Multiple agents.

Abstract: This paper proposes a bio-inspired framework for adapting 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 well-developed. The framework was constructed as a general-purpose middleware system and allowed us to define agents as Java objects. We present several evaluations of the framework in a distributed system instead of any simulation-based systems.

SCIENCE AND TECHNOLOGY PUBLICATIONS

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. We construct a framework for building and operating distributed applications with the notion of cellular differentiation and division in cellular slime molds, e.g., dictyostelium discoideum and mycelium. 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. The framework enables software components, called agents, to differentiate their functions according to their roles in whole applications and resource availability, as just like cells. It involves treating the undertaking/delegation of functions in agents from/to other agents as their differentiation factors. 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

2 RELATED WORK

The notion of self-organization is rapidly gaining importance in the area of distributed systems. We discuss several related studies on software adaptation in distributed systems.

One of the most typical self-organization approaches to distributed systems is swarm intelligence (Bonabeau et al., 1999; Dorigo and Stutzle, 2004). Although there is no centralized control structure dictating how individual agents should behave, interactions between simple agents with static rules often lead to the emergence of intelligent global behavior. There have been many attempts to apply self-organization into distributed systems, e.g., a myconet model for peer-to-peer network (Snyder et al., 2007), and a cost-sensitive graph structure for coordinated replica placement (Herrman, 2007). Most existing approaches only focus on their target problems or applications but are not general purpose, whereas distributed systems have a general-purpose infrastructure. Our software adaptation approach should be independent of applications. Furthermore, most existing self-organization approaches explicitly or implicitly assume a large population of agents or boids. However, 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 agents.

One of the most typical approaches to self-organization is genetic programming (Koza, 1992).

Satoh I..
CELLULAR DIFFERENTIATION-BASED APPROACH FOR DISTRIBUTED SYSTEMS.
DOI: 10.5220/0003642701310136
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2011), pages 131-136
ISBN: 978-989-8425-83-6
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

The fitness of every individual program in the population need to be evaluated in each generation and multiple individuals are stochastically selected from the current population based on their fitness. However, since distributed systems may have an effect on the real world and be used for mission-critical processing, there is no chance of ascertaining the fitness of randomly generated programs. Our framework should be conservative rather than emergent in the sense that adaptation caused by the framework must be within our prior expectation for reasons of reliability and availability.

Many simulation-based approaches for self-organization for distributed systems have been explored. However, there is a serious gap between real distributed systems and those that are simulation-based. In fact, a real distributed system is just a complex system so that it is difficult to model or simulate the system itself.[1]

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. (Bigus et al., 2002) proposed an agent-based toolkit for autonomic systems, where each agent has a closed-loop 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. Jaeger et al. (Jaeger et al., 2007) introduced the notion of self-organization to ORB and a publish/subscribe system. Holvoet et al. (Holvoet et al., 2009) supported self-organizing coordination between agents. These existing studies could select and invoke software components according to their context, but they could not adapt software components themselves. Georgiadis et al. (Georgiadis et al., 2002) 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. (Cheng et al., 2006) 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.

[1]We do not intend to deny simulation-based approaches. Nevertheless, we need a basic model, including parameters, for real distributed systems before simulating such systems.

We proposed a nature-inspired approach to dynamically deploying agents at computers in our previous papers (Satoh, 2007; Satoh, 2008). 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 BASIC APPROACH

This paper introduces the notion of (de)differentiation into a distributed system as a mechanism for adapting software components, which may be running on different computers connected through a network.

Differentiation. When dictyostelium discoideum cells aggregate, they can be differentiated into two types: prespore cells and prestalk cells. Each cell tries to become a prespore cell and periodically secretes cAMP to other cells. If a cell can receive more than a specified amount of cAMP from other cells, it can become a prespore cell. There are three rules. 1) cAMP chemotaxically leads other cells to prestalk cells. 2) A cell that is becoming a prespore cell can secrete a large amount of cAMP to other cells. 3) When a cell receives more cAMP from other cells, it can secrete less cAMP to other cells.

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. Each agent initially intends to progress all its functions and periodically multicasts *restraining* messages to other agents federated with it. Restraining messages lead other agents to degenerate their functions specified in the messages and to decrease the superiority of the functions. As a result, agents complement other agents in the sense that each agent can provide some functions to other agents and delegate other functions to other agents that can provide the functions.

Dedifferentiation. 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 less-developed or inactive functions again.

Remarks. Each specialized cell type in an organism expresses a subset of all the genes that constitute the genome of that species. Each cell type is defined by its particular pattern of regulated gene expression. With a few exceptions, differentiation does not involves change in the DNA sequence itself. That is to say, different cells may have different physical characteristics, but they have almost the same gene and this is largely due to highly-controlled modifications in their gene expression. Each agent can explicitly preserve programs for defining all its functions after it has been differentiated.

4 DESIGN AND IMPLEMENTATION

The framework was constructed as a general-purpose middleware system and allowed us to define agents as Java objects. The whole system consists of two parts: runtime systems and agents. The former is a middleware system for running at computers and the latter is a self-contained and autonomous entity.

4.1 Agent

Each agent consists more than one application-specific function, called *behavior* part, and its state, called *body* part, with information for differentiation, called *attribute* part. The *body* part is responsible for maintaining program variables shared by its behaviors parts. When it receives a request message from the external system or other agents, it dispatches the message to the behavior part that can handle the message. The *attribute* part maintains descriptive information with regard to the agent, including its own identifier.

4.2 Differentiation

Behaviors in a agent, which are delegated from other agents more times, are well developed, whereas other behaviors, which are delegated from other agents less times, in the cell are less developed. Finally, the agent only provides the former behaviors and delegates the latter behaviors to other agents. Each agent (k-th) assigns its own maximum to the total of the weights of all its behaviors. The agent has behaviors $b_1^k, \ldots, b_n^k$, w_i^k is the weight of behavior b_i^k. W_i^k is the maximum of the weight of behavior b_i^k. The maximum of the total of the weights of its behaviors in k-th agent must to be less than W^k. ($W^k \geq \sum_{i=1}^{n} w_i^k$), where $w_j^k - 1$ is 0 if w_j^k 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. Our mechanism consists of two phases. The first-step phase involves the progression of behaviors.

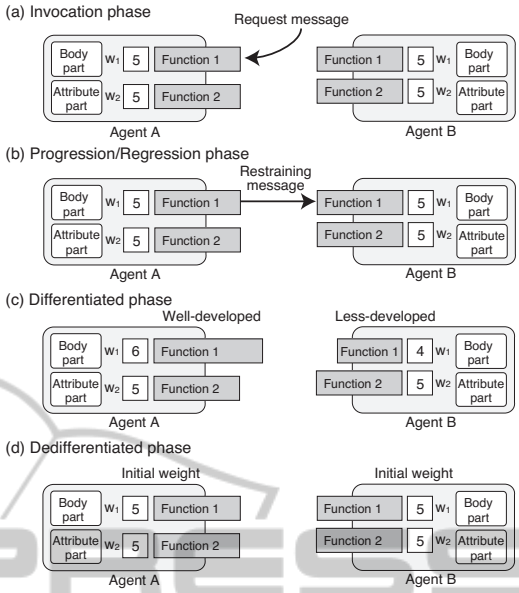

Figure 1: Differentiation mechanism for agent.

Step 1. When an agent (k-th agent) receives a request message from another agent, it selects the behavior (b_i^k) that can handle the message from its behavior part and dispatches the message to the selected behavior (Figure 1 (a)).

Step 2. It executes the behavior (b_i^k) and returns the result.

Step 3. It increases the weight of the behavior w_i^k.

Step 4. It multicasts a restraining message with the signature of the behavior, its identifier (k), and the behavior's weight (w_i^k) to other agents (Figure 1 (b)).

Note that, when behaviors are invoked by their agents, their weights are not increased. The key idea behind this approach is to distinguish between internal and external requests. If the total of the weights of the agent's behaviors, $\sum w_i^k$, is equal to their maximal total weight W^k, it decreases one of the minimal (and positive) weights (w_j^k is replaced by $w_j^k - 1$ where $w_j^k = \mathbf{min}(w_1^k, \ldots, w_n^k)$ and $w_j^k \geq 0$). The above phase corresponds to the degeneration of agents. Restraining messages correspond to cAMP in differentiation.[2]

[2]When the runtime system multicasts information about the signature of a behavior in restraining messages, the signature is encoded into a hash code by using Java' serial versioning mechanism and transmitted as the code.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

The second-step phase supports the retrogression of behaviors.

Step 1. When an agent (k-th agent) receives a restraining message with regard to b_i^j from another agent (j-th) , it looks for the behaviors ($b_m^k, \ldots b_l^k$) that can satisfy the signature specified in the receiving message.

Step 2. If it has such behaviors, it decreases their weights ($w_m^k, \ldots w_l^k$) in its first database and updates the weight (w_i^j) in its second database (Figure 1 (c)).

Step 3. If the weights ($w_m^k, \ldots, w_l^k$) are under a specified value, e.g., 0, the behaviors ($b_m^k, \ldots b_l^k$) are inactivated.

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

Step 1. When an agent (k-th agent) wants to execute a behavior b_i , it looks up the weight (w_i^k) of the same or compatible behavior from its first and database and the weights ($w_i^j, \ldots, w_i^m$) of such behaviors ($b_i^j, \ldots, b_i^m$) from the second database.[3]

Step 2. If multiple agents, including itself, can provide the wanted behavior, it selects one of the agents according to selection function ϕ^k, which maps from w_i^k and $w_i^j, \ldots, w_i^m$ to b_i^l, where l is k or $j, \ldots, m$.

Step 3. It delegates the selected agent to execute the behavior and waits for the result from the agent.

The approach permits agents to use their own evaluation functions, ϕ, because the selection of behaviors often depends on their applications. Although there is no universal selection function for mapping from behaviors' weights to at most one appropriate behavior like a variety of creatures, we provide several functions. For example, one of the simplest evaluation functions makes the agent that wants to execute a behavior select the behavior whose weight has the highest value and signature matches the wanting behavior if its first and second databases recognizes one or more agents that provide the same behavior, including itself. Since each agent records the time behaviors are invoked and are received the results, it selects behaviors provided in other agents according to the average or worst response time in the previous processing.

[3]The agent (k-th) may have more than one same or compatible behavior.

4.3 Dedifferentiation

Each agent (j-th) periodically multicasts messages, called *heartbeat messages*, for behavior (b_i^j), which is still activated with its identifier (j) via the runtime system. This involves one of either the following cases.

Case 1. When an agent (k-th) receives a heartbeat message with regard to behavior (b_i^j) from another agent (j-th), it keeps the weight (w_i^j) of the behavior (b_i^j) in its second database.

Case 2. When an agent (k-th) does not receive any heartbeat messages with regard to behavior (b_i^j) from another agent (j-th) for a specified time, it automatically decreases the weight (w_i^j) of the behavior (b_i^j) in its second database, and resets the weight (w_i^k) of the behavior (b_i^k) to be initial value or increases the weight (w_i^k) in its first database (Figure 1 (d)).

Note that the behavior b_i^k is provided by k-th agent and the behavior b_i^j is provided by j-th agent. The weights of behaviors provided by other agents are automatically decreased without any heartbeat messages from the agents. Therefore, when an agent terminates or fails, other agents decrease the weights of the behaviors provided from the agent and then if they have the same or compatible behaviors, they can activate the behaviors, which may be inactivated.

After sending a request message is sent to another agent, if the agent waits for the result to arrive longer than a specified time, it selects one of the agents that can handle the message from its first and second databases 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 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 11 ms. The cost of transmitting a request message between two computers was 22 ms through TCP. These costs were estimated from the measure-

ments 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.

5.1 Experiments: Differentiation

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 200 ms 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 200 ms, 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.

5.2 Experiments: Dedifferentiation

The second experiment was carried out to evaluate

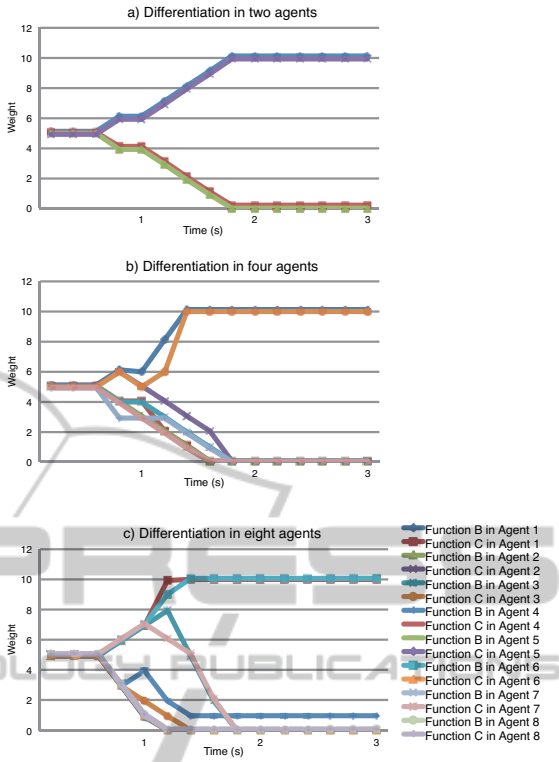

Figure 2: Degree of progress in differentiation-based adaptation.

the ability of the agents to adapt to two types of failures in a distributed system. 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_B^i, w_C^i, w_D^i) in i-th agent were $(10,0,0)$ in the first, $(0,10,0)$ in the second, and $(0,0,10)$ in the third.

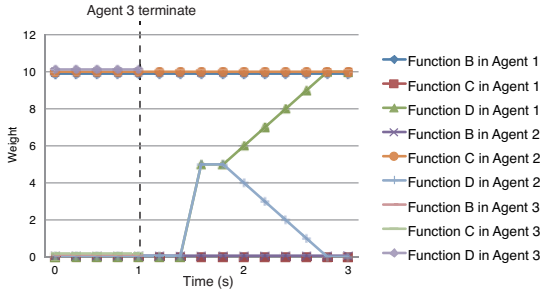

Figure 3: Degree of progress in adaptation to failed agent.

6 CONCLUSIONS

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.

In concluding, we would like to identify further issues that need to be resolved. We did not intend to use any simulation-based approaches, because the performance of software adaptation on distributed systems greatly depends on the systems and the demands of their applications. It is almost impossible to simulate such systems accurately. After we evaluate software adaptation with this framework on real distributed systems, we plan to construct a simulation system based on the results. Our software adaptation mechanism depends on selection functions, but from our evaluations we knew that there was no universal function. Nevertheless, we plan to refine and extend selection functions.

ACKNOWLEDGEMENTS

This research was supported in part by a grant from the Promotion program for Reducing global Environmental loaD through ICT innovation (PREDICT) made by the Ministry of Internal Affairs and Communications in Japan.

REFERENCES

Bigus, J., Schlosnagle, D., Pilgrim, J., Mills, W., and Diao, Y. (2002). Able: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, 41(3):350–371.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

Cheng, S., Garlan, D., and Schmerl, B. (2006). Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of International Workshop on Self-adaptation and Self-managing Systems (SEAMS'2006)*, pages 2–8. ACM Press.

Dorigo, M. and Stutzle, T. (2004). *Ant Colony Optimization*. MIT Press.

Georgiadis, I., Magee, J., and Kramer, J. (2002). Self-organising software architectures for distributed systems. In *Proceedings of 1st Workshop on Self-healing systems (WOSS'2002)*, pages 33–38. ACM Press.

Herrman, K. (2007). Self-organizing replica placement - a case study on emergence. In *Proceedings of 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'2007)*, pages 13–22. IEEE Computer Society.

Holvoet, T., Weyns, D., and Valckenaers, P. (2009). Patterns of delegate mas. In *Proceedings of 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'2009)*, pages 1–9. IEEE Computer Society.

Jaeger, M. A., Parzyjegla, H., Muhl, G., and Herrmann, K. (2007). Self-organizing broker topologies for publish/subscribe systems. In *Proceedings of ACM symposium on Applied Computing (SAC'2007)*, pages 543–550. ACM Press.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Satoh, I. (2007). 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)*, volume 4415 of *Lecture Notes in Computer Science (LNCS)*, pages 185–198. Springer.

Satoh, I. (2008). Test-bed platform for bio-inspired distributed systems. In *Proceesings of 3rd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*.

Snyder, P. L., Greenstadt, R., and Valetto, G. (2007). 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)*, pages 40–50. IEEE Computer Society.