Autonomous Function Composition for Information Network Adaptive to Changes in Service Request

Shunji Okamoto and Naoki Wakamiya
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, Japan
s-okamot@ist.osaka-u.ac.jp, wakamiya@ist.osaka-u.ac.jp

ABSTRACT
Information networks have been introducing new or improved protocols one after another to satisfy diverse requirements of a variety of emerging applications. Consequently, current network systems have become considerably large, complex, and even fragile. In this paper, to accomplish a sustainable network system which autonomously adapts to diverse requests by dynamically generating necessary control mechanisms, we propose an autonomous function composition method. More specifically, each node has small pieces of networking functions, called function modules, and appropriately combines them to answer each service request. Combinations are refined taking into account the degree of satisfaction of users by a genetic algorithm. Through simulation experiments, we verify that our proposal can adapt to changes in requests in the multi-user environment.

Categories and Subject Descriptors
C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design

General Terms
Algorithms, Design

Keywords
Functional Evolution, Adaptation, Genetic Algorithm

1. INTRODUCTION
Information networks are now an indispensable social infrastructure which accommodates a variety of traffic generated by diverse applications. With the emergence of new applications and communication services, network systems have been growing by introducing new or improved networking technologies one after another. As a consequence of such ad-hoc and patching-based development, current network systems have become functionally large and complex, which makes network systems fragile and uncontrollable. To tackle the problem, there have been several attempts on development of sustainable network systems which are adaptive and evolvable. Under the concept of Autonomic Networking, researches such as functional evolution of network inspired by evolution of organisms and embryo formation [1] and flexible service composition inspired by metabolic reactions [2] are conducted. Furthermore, it is considered to modularize networking services and functions into functional blocks or functional elements and combine them to generate a desired function in Autonomic Network [3] and Fraglet [4]. In addition, in [4], [5], [6], they adopt evolutionary algorithms to evolve functions or functional elements.

In this paper, we take the similar approach but propose more detailed algorithm. As a basis of our research, we consider that sustainable development of information networks which is self-adaptive and self-evolvable can be realized by modularizing networking functions, combining function modules in response to a service request of a user, evolving them in accordance with dynamically changing requests. For this purpose, we propose an autonomous function composition method. More specifically, a node has a table of combinations of function modules deposited in a local pool. An entry of the table consists of a combination and a score reflecting the degree of satisfaction of a user with the combination. By using scores, a node stochastically selects a combination to apply to a request and conducts a genetic algorithm-based adaptation of combinations.

2. AUTONOMOUS AND ADAPTIVE FUNCTION COMPOSITION
In this section, we first give an outline of our proposal and then describe details.

2.1 Outline of Proposal
In the proposed method, a node dynamically composes a function by combining appropriate function modules (Fig. 1.). We assume that a function module is a tuple like Fraglet [4] that specifies a single processing or a program code for a single function element. There could be multiple function modules of the same function element, but they are different in parameters, algorithms, or mechanisms. We denote functionally different modules as, for example, Φ_1 and Ψ_1 which correspond to different function elements Φ and Ψ, respectively. On the contrary, function modules Φ_1 and Φ_2 realize the same function element Φ, but they have partial differences. As a simple example, a general CSMA/CD consists of four function elements, i.e. carrier sense, back off,

BICT 2014, December 01-03, Boston, United States

DOI 10.4108/icst.bict.2014.257879

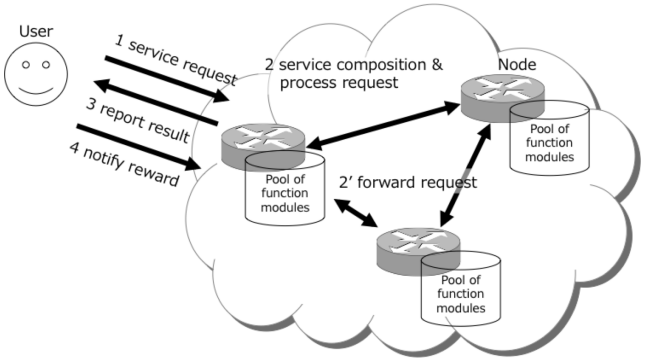

Figure 1: Autonomous function composition

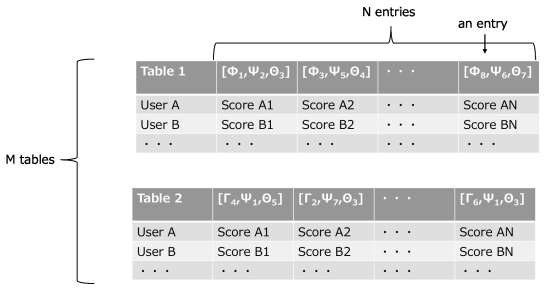

Figure 2: Combination tables

transmission, and collision detection. A node would have several function modules of carrier sense different in carrier sense threshold. Regarding back off, an initial timer value and an exponent are different among function modules.

A user, which does not necessarily mean a human but includes an application program, a control program, and a node, requests a service or a networking function by sending a request message to a node. A request message consists of identifiers of function modules and data to process. A request message would define a set of specific functions, e.g. $[\Phi_1, \Psi_2, \Theta_3]$. A wild card can be used as $[\Phi_*, \Psi_2, \Theta_3]$ and $[*, \Psi_2, \Theta_3]$. We should note here that an order has a meaning, thus $[\Phi_1, \Psi_2, \Theta_3]$ and $[\Psi_2, \Phi_1, \Theta_3]$ are different.

On receiving a request message, a node first checks whether all specified function modules are in a pool. If there is any missing module, a node forwards the request message to an adjacent node. Otherwise, a node determines function modules to combine and processes the message. For a function element specified by a wild card, a node selects a function module stochastically. So that a node can effectively and efficiently select an appropriate combination of function modules against an ambiguous request, it has a combination table and selection is performed based on the table. A combination table is updated by using a genetic algorithm to prepare and maintain an appropriate combination in accordance with dynamically changing and diverse requests from users. After processing a request message, a result is reported to a user. Then, the user informs of the degree of satisfaction with the result, which is used in combination selection and table updating.

In this paper, we enable functional evolution of an information network by a genetic algorithm, but evolvability is limited to combination of function modules. Therefore we additionally need a mechanism to introduce a new function module. For efficient service provision, autonomous mechanisms to distribute function modules and navigate a request message to an appropriate node. However, they are out of scope of this paper and remain future work.

2.2 Structure of combination table

A node has M combination tables each of which corresponds to different combinations of function elements such as $[\Phi, \Psi, \Theta]$, $[\Gamma, \Psi, \Theta]$, $[\Psi, \Theta, \Gamma]$, and $[\Phi, \Psi, \Theta, \Omega]$ as illustrated in Fig. 2. Each combination tables consists of N entries of the same combination but with different parameters. Each entry has a list of scores for a set U of users. Scores are used for selection and evolution of combination. In the following, a score of combination i $(1 \leq i \leq N)$ for user u $(u \in U)$ in a combination table is denoted as $score_{u,i}$ $(s_{min} \leq score_{u,i})$, where s_{min} is a small positive real number.

2.3 Selection of combination

When a node receives a request message, it first checks the existence of requested function modules in a pool. If there is any missing module, it forwards the request message to an adjacent node. Next, if a node does not have a combination table of a requested combination, it generates a new one. For unspecified function modules and parameters in an ambiguous request, e.g. $[\Phi_*, \Psi_2, \Theta_3]$ and $[*, \Psi_2, \Theta_3]$, a node selects function modules in a uniform random manner to make an initial table. A combination table is discarded, when it is not used for a certain duration of time. If there are more than one combination table corresponding to an ambiguous request, a node selects a table with the smallest number of users. Then, a user is added to a combination table if it does not exist. Scores for a new user are set at averages of existing users. Finally, a node selects a combination by using a generated or selected combination table and applies combined function modules to a request.

A node updates a combination table by a genetic algorithm every Q requests. When a new combination is generated, a node applies the combination to the following P $(P < Q)$ requests and selects a combination based on scores for the remaining $Q - P$ requests. Otherwise, a node applies a combination selected based on scores to all Q requests. Such deterministic selection of a new combination is necessary to avoid immediate death of a new combination by accumulating scores which reflects rewards from users. In the case of stochastic selection, probability $P(u, i)$ with which a node selects combination i for user u $(u \in U)$ is defined as $P(u,i) = \frac{score_{u,i}}{\sum_{k=1}^{N} score_{u,k}}$.

A node reports a result of message processing to a requesting user. The user notifies the node of the degree of satisfaction in the form of a scalar value which we call reward $(0 \leq reward \leq 1)$. A node updates the score as $score_{u,i} \leftarrow score_{u,i} \times \alpha + reward_{u,i}^{\beta} \times (1 - \alpha)$. Here, α $(0 \leq \alpha \leq 1)$ is a smoothing coefficient and β $(1 \leq \beta)$ is a parameter that defines the effect of the reward to the score.

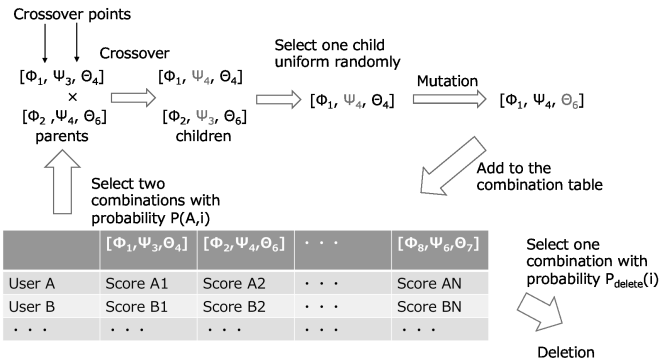

Figure 3: Updating combination table

2.4 Updating combination table

We regard a combination of function modules as a chromosome and a function module as a locus or a gene in applying a genetic algorithm to adaptive evolution of a combination table. As illustrated in Fig. 3, a node first selects two combinations as parents from a combination table by using the probability distribution $P(u,i)$ of randomly selected user u. The same combination can be selected as parents. Next one combination is selected as candidates of deletion by using $P_{delete}(i) = \frac{\frac{1}{\sum_{u\in U} P(u,i)}}{\sum_{k=1}^{N} \frac{1}{\sum_{u\in U} P(u,k)}}$. Then, a node generates two children of parents by exchanging function modules between two randomly selected locations between parents, that is, the two-point crossover method. One child is randomly selected and discarded. The remaining child undergoes mutation with the fixed probability m, where a uniformly randomly selected function module of a child is replaced with a function module uniformly randomly selected from a pool. If a child has already existed in a combination table, the child is deleted. If a child is a new combination, a combination selected for deletion is replaced with the child. Scores of a child are set at s_{min}.

3. EVALUATION

We evaluate the proposal by simulation experiments using multi-user scenarios.

3.1 Simulation setting and measures

We use NetLogo 5.0.4 for simulation. In this paper, to confirm the effectiveness of our function composition method, we only consider the case of a single node and a single combination table ($M = 1$) of 5 function elements. The size N of a table is 100 and each function elements has 100 variants of function modules. In total a node has 500 function modules in a pool. All users send the same ambiguous requests, e.g. $[\Phi_*, \Psi_*, \Theta_*, \Omega_*, \Lambda_*]$. For a purpose of evaluation, we assume that we can define the distance between function modules of the same function element. For example, the distance between Φ_j and Φ_{j+1} $(0 \leq j < 99)$ is 1.

We consider the reward is determined based on the distance. We define $\mu_{u,f}$ $(0 \leq \mu_{u,f} \leq 99)$, called a request top, as the type of function module of function element f, with which user u $(u \in U)$ is fully satisfied. The reward of combination i with function module of type f, is derived as $s_u(f_i) = \exp\left(-\frac{(f_i-\mu_{u,f})^2}{2\sigma_{u,f}^2}\right)$. $\sigma_{u,f}^2$ defines the range where the reward is given. Then, the reward of combination i is defined by the average of rewards as $r_u(i) = \frac{\sum_{f\in R} s_u(f_i)}{|R|}$. R is a set of function modules specified in a request. Our proposal is applicable to cases of different reward functions, e.g. $s_u(f_i) = \frac{1}{1+\exp(\frac{-gf_i}{99})}$ or $s_u(f_i) = 1$ for $f_i = \mu_{u,f}$, and we plan to investigate their influence in future evaluation.

We evaluate our proposal from a viewpoint of degree of satisfaction of users, more specifically, the expected reward. The expected reward for function module f from user u is derived as $E_f(u) = \sum_{i\in F} r_{u,i} \times P(u,i)$. F is a set of combinations having a function module of function element f. Then, the expected reward regarding user u is derived as $E(u) = \frac{\sum_{f\in R} E_f(u)}{|R|}$. We call $E(u)$ fitness. We consider that users send a request one after another. We call a duration in which every user sends one request to a node a round. Users have different request tops as will be explained later, whereas we set $\sigma_{u,f}^2 = 5$ for all users.

At the beginning of a simulation run, a node has unduplicated N combinations of function modules each of which is randomly chosen from 100 variants. The initial score is identically set at $s_{min} = 1.4e-45$. Regarding parameters of our proposal, we empirically set $P = |U|$, $Q = P \times 2$, $m = 5\%$, $\alpha = 1/8$, and $\beta = 4$. In the next section, we show results averaged over 50 simulation runs.

3.2 Evaluation results and Discussion

First, we consider a change in a request top (case 1). There are four users numbered from 1 to 4. They have different request tops $\mu_{u,f} = 20$, 40, 60, and 80 independently of function element f, respectively. At 12000th round, user 2 and user 3 change their request tops $\mu_{u,f}$ from 40 to 0 and from 60 to 99, respectively. Time variations of fitness are shown in Fig. 4. Figure 5 shows frequency distributions of function module, which is the ratio of combinations containing the module in N combinations. As shown in the figures, users who do not change their request tops are not affected by the behavior of users 2 and 3. Although a combination table is shared among users, combinations fitting to those users have high scores and are less likely to be selected for deletion.

On the contrary, fitness of user 2 and user 3 once drops to about zero when they change request tops, because a combination table lacks combinations fitting to their new requests. Right after the change, combinations around 40 and 60 are applied to each of their requests because score are high. However, such combinations do not receive sufficient rewards and as a result their scores decrease. It increases the relative possibility of selection of combinations around 0 and 99 and their scores eventually increase. It further enables generation of new combinations around 0 and 99 without discarding combinations fitting to the other users. Consequently, as shown in Fig. 5, combinations around 40 and 60 disappear and those around 0 and 99 emerge. In conclusion, our proposal can adapt to multiple changes in requests.

Next, we consider addition of new users (case 2). Initially,

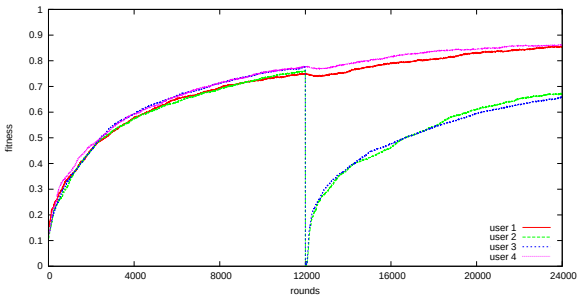

Figure 4: Time variation of fitness (case 1)

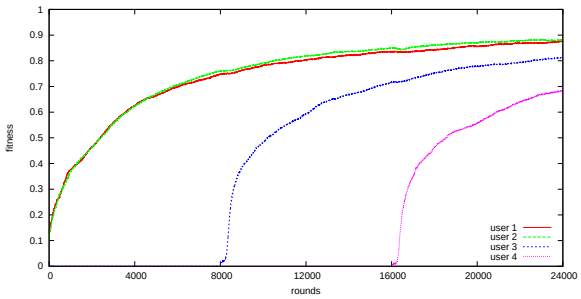

Figure 6: Time variation of fitness (case 2)

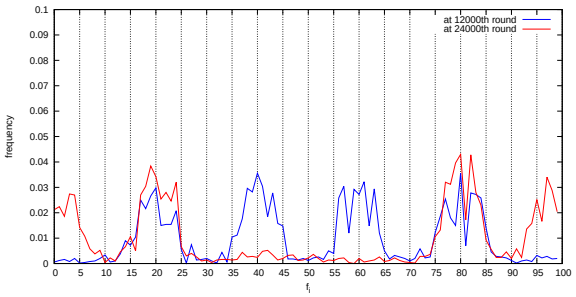

Figure 5: Distribution of function elements (case 1)

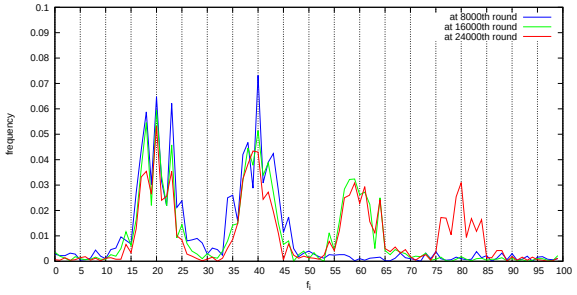

Figure 7: Distribution of function elements (case 2)

there are two users with request tops $\mu_{u,f} = 20$ and 40, respectively. At 8000th round, a new user 3 with a request top of 60 joins. Then, at 16000th round, another user 4 with a request top of 80 joins. Results are shown in Figs. 6 and 7. Since a combination table is shared among users, as shown in Fig. 7, distribution of combinations around 20 and 40, i.e. request tops of existing users, decreases as new users join and combinations around 60 and 80 emerge. However, Fig. 6 shows that fitness of new users gradually increase without affecting fitness of existing users. Therefore, our proposal is adaptive to emergence of new requests.

4. CONCLUSION

In this paper, we propose an autonomous function composition method that provides users with services by dynamically combining function modules. To accomplish functional evolution of information networks, combinations of function modules are adapted to changes in requests by using a genetic algorithm. Through simulation evaluation, we verify that our proposal can adapt to changes in requests in the multi-user environment.

As future work, we need to consider the influence of parameters such as P, Q, α, β, and m, on speed of evolution and fine-tuning of combinations. We also plan to introduce additional mechanisms such as distribution of service modules and navigation of request messages.

5. ACKNOWLEDGMENTS

This work was partly supported by Grant-in-Aid for Scientific Research(B) 25280029 of Japan Society for the Promotion of Science.

6. REFERENCES

[1] D. Miorandi, L. Yamamoto, and F. D. Pellegrini. A survey of evolutionary and embryogenic approaches to autonomic networking. *Computer Networks*, 54(6):944–959, 2010.

[2] M. Viroli and M. Casadei. Chemical-inspired self-composition of competing services. In *Proceedings of ACM Symposium on Applied Computing (SAC '10)*, pages 2029–2036, March 2010.

[3] M. Sifalakis, A. Louca, G. Bouabene, M. Fry, A. Mauthe, and D. Hutchison. Functional composition in future networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 55(4):987–998, March 2011.

[4] C. F. Tschudin. Fraglets - a metabolistic execution model for communication protocols. In *Proceedings of Annual Symposium on Autonomous Intelligent Networks and Systems (AINS)*, July 2003.

[5] T. Nakano and T. Suda. Self-organizing network services with evolutionary adaptation. *IEEE Transactions on Neural Networks*, 16(5):1269–1278, 2005.

[6] L. Yamamoto and C. Tschudin, editors. *Genetic Evolution of Protocol Implementations and Congfiurations*, 2005.