Dynamic Control and Construction Method for Multiagent Systems Based on an Evolutional Agent System

Akiko Takahashi[1] and Tetsuo Kinoshita[2]

[1]*Sendai National College of Technology*
[2]*Research Institute of Electrical Communication, Tohoku University*
akiko@sendai-nct.ac.jp, kino@riec.tohoku.ac.jp

Abstract

We propose a novel multiagent system for controlling multiagent service-provisioning systems while continuing to provide service. The new method uses "control possibility" and "control effect" to reduce the degradation of Quality of Service to the absolute minimum. We evaluate the effectiveness of the new method by implementing it in a multimedia communication system. The results show that the proposed system can provide better multimedia communication services compared with a conventional multimedia communication system.

Keywords: *system margin; QoS; multiagent system; multimedia communication system; service composition; control possibility; control effect*

1. Introduction

Operating environments for information and communication services include desktop PCs, mobile computers, smartphones, broadband LAN, and wireless LAN. These platform and network environments change randomly with changes in applications, hardware, and other factors. To respond to these changes and maintain adequate service, most users must construct and control the systems themselves. However, control of such complex systems requires specialized experience that most users do not have [1, 2].

We seek an optimal situation in which network services are maintained with minimum burden on a typical user; specifically, we seek an "autonomous service-provision system." In general, it is difficult to avoid some system instabilities with accompanying decreases in Quality of Service (QoS), but such losses can be minimized. For example, existing autonomous service-provision systems are controlled and constructed to restore QoS after detecting a decrease in QoS, but they cannot always actually restore QoS [3].

In this paper, we propose a novel method for proactively controlling multiagent systems to reduce the decrease in QoS from a service-provisioning system. The intent is to maintain satisfactory service in response to irregular changes in user requirements, platform environments, and network environments. We define "control possibility" and "control effect" against QoS changes during service provision as a "system margin" based on an "Evolutional Agent System" (EAS). Using these, we control the system to realize a method for providing satisfactory service. Moreover, to evaluate its effectiveness, we implement the proposed control system in a multimedia communication system. The results show that the

new system can provide better multimedia communication services compared with a conventional multimedia communication system. This is because systems based on conventional methods control themselves *after* detecting undesirable resource conditions related to QoS.

The remainder of this paper is organized as follows. We describe related work and our proposal in Section 2. Then we propose EAS and the system margin in Section 3. In Section 4, we describe evaluation along with the design and implementation of a multimedia communication system based on system margin. Finally, Section 5 summarizes our conclusions.

2. Related Work and Proposal

2.1. Related work

In autonomous service-provision systems, users must construct and control the service-provisioning system dynamically to maintain adequate service. But for some service-provision systems based on multiagent systems, methods have been reported [4, 5] for constructing the overall system by combining software agents that work autonomously. During service provisioning, these systems change parameter values and reorganize the agents when the environment changes or when QoS otherwise degrades. Autonomous service-provision systems based on these methods are scalable and flexible because the agents need not consider the behavior of the overall system when individual members of the system are added, replaced, or fail. Moreover, users are not required to perform complex tasks to maintain service because agents construct the system to detect changes in environmental conditions and to respond to those changes. Nevertheless, it is difficult to avoid temporary undesirable situations because these methods cannot consider the effects of control and reorganization [1].

One report describes an agent organization model to construct a system that autonomously recovers from undesirable conditions when trouble occurs [6]. This model has design member agents to control the agent organization that monitors the condition of the overall agent system. Through the autonomous behavior of the member agents based on the model, the agent system is constructed systematically. During service provision, if some agent is aborted or some trouble occurs, which does not satisfy the system purpose, then the agent organization is reconstructed. Other models of multiagent systems have also been proposed. One is an organizational model that defines the knowledge of the system's organizational structure and capabilities to enable dynamic system reorganization [7]. Another is a mathematical model prescribing that the agents act by Markov chains to describe a system's emergent behavior as a Markov Decision Process [8]. These methods can recover a system autonomously; however, in recovering a system, it is difficult for them to avoid some suspension in service.

Moreover, a multiagent system architecture that contains a policy to manage applications, resources, and service provision has been proposed [9]. The policy leads to functions that can construct systems on the basis of user requirements, recover from some troubles, and optimize the system. However, the policy to recover from troubles restricts applicable services and is redundant. Moreover, if the same policy is applied to every system component, the overall system becomes overloaded.

In general, a multiagent system requires a self-organizing model and a self-control model. Although self-organizing models have been proposed [10–12], they specialize in constructing the agent organization. For self-control models [13, 14], the candidates are the control scheme or the trust assurance of the overall agent system. However, if the model is to realize a

manageable, systematic, and reliable multiagent system, the design model for a multiagent system must consider both a self-organizing model and a self-control model.

In the design models of previous studies, a semantic model for specifying and reasoning about components of open distributed systems has been proposed [15]. In this model, multimedia actor behaviors satisfy the specified requirements and provide the required multimedia service. The behavior specification leaves open the possibility of various algorithms for resource management. In other work, a quantitative performance model has been reported to predict the performance of a component-based server-side application in the design phase [16]. The model requires input from an application-independent performance profile of the underlying component technology platform; the model also needs a design description of the application. The results from the model allow the architect to make early decisions between alternative application architectures in terms of their performance and scalability. However, the above issues remain obstacles to realizing flexible QoS control functions for some network service systems that are designed and implemented as multiagent systems.

In general, conventional methods have difficulty avoiding temporary QoS degradation or incomplete control because such methods control themselves after detecting undesirable situations that occur while providing service to users.

2.2. Challenge and proposal

To restore a service-provisioning system from an undesirable situation, conventional methods control the system after detecting QoS degradation. However, some conventional methods cannot avoid temporary degradation of QoS or interruption of service, while others cannot always manage to actually restore QoS. In this paper, we focus on methods for inhibiting undesirable conditions (conditions that do not satisfy user requirements) and controlling against QoS degradation by preparing for changes in resource status.

To date, it remains difficult for service-provisioning systems to assess the changes in resources that cause undesirable conditions. Therefore, challenges remain for realizing an autonomous service-provision system that controls itself, prepares for changes, and considers the effects of those controls.

In this paper, we propose a policy for preparing for undesirable conditions in advance in order to control a multiagent system. The policy is tolerant of changes in resource status and performance characteristics of agents, which are the candidates of reorganization. It actively reconstructs a system when the system cannot tune parameters sufficiently to restore QoS in response to changes in resource status. Challenges inherent to realizing such a system include the following.

(P1) Estimation of QoS after parameter tuning: the policy must estimate whether the system can be controlled and whether the control can provide sufficient QoS while providing services.
(P2) Estimation of QoS after reconstruction: the policy must assess the reconstruction plan and its effects.

For these challenges, we propose a new policy of performance characteristics: the “system margin.” This new policy estimates the possibility of control and their effects.

(S1) Margin of tuning: tolerance of systems to changes in resource status. This defines the “control possibility” and the degree of QoS provided to users by tuning. If it is difficult to maintain sufficient QoS, an advanced reconstruction is considered.

(S2) Margin of alternative: the effects of system reconstruction. This defines the "control effect" and the expected QoS provided by the candidate for the alternative agents in reconstruction. A system based on this policy always includes reconstruction.

3. Control Method based on the System Margin

3.1. Evolutional Agent System

We have designed a control method based on the system margin using an EAS: a multiagent system that controls and reconstructs itself actively to recover QoS. The EAS analyzes the system environment rather than merely responding passively to change. Specifically, the EAS controls itself actively when another system can provide better QoS and functions than the system that is currently providing the service. For example, control is provided when an agent organization exists that can improve QoS and characteristics of agent behaviors. Therefore, in EAS, an evolutional mechanism exists to improve system performance while analyzing behavioral characteristics and environment information. We designed the system margin as an EAS mechanism.

3.2. Model of the Evolutional Agent System

This section describes an EAS in a syntactic manner. An EAS consists of EAS names, requirements, EAS structure, functions, environment, and properties.

[Definition 1] An EAS is designed on the basis of *ne*, *R*, *S*, *F*, *E*, and *P*:

$$EAS = <ne, R, S, F, E, P>$$

Here, *ne* is the name of the EAS, *R* is a set of requirements given to the EAS, *S* is the structure of the EAS, *F* is a set of functions working in the EAS, *E* is the environment in which the EAS operates, and *P* is the EAS property. □

Detailed elements of the EAS are defined next.

[Definition 2] *R* contains req_k (k = 1, 2, …, *N-r*), where req_k is the description of a required service function $req\text{-}f_k$ and $req\text{-}f\text{-}q_k$ is the required quality of $req\text{-}f_k$,

$$R = \{req_k \mid req_k = <req\text{-}f_k, req\text{-}f\text{-}q_k> ; k=1, .., N\text{-}r \}.$$

Here, *N-r* is the total number of required functions. □

[Definition 3] *S* is an organization of the EAS. It consists of *AG*, *STR*, and *AL-AG*,

$$S = <AG, STR, AL\text{-}AG>.$$

Here, *AG* is a set of member agents of the EAS, *STR* is the structure of an agent organization, and *AL-AG* is a set of alternative agents that can be replaced with a member of *AG*. □

AG contains ag_i (i = 1, .., *N-a*), where an ag_i is determined with the identifier na_i, a set of received messages $M\text{-}ag_i$, a set of services that can be provided by ag_i $Sv\text{-}ag_i$, a set of action knowledge $K\text{-}ag_i$, processing load of ag_i $w\text{-}ag_i$, and the communication agent set CoA_i:

$AG = \{ag_i \mid ag_i = <na_i, M\text{-}ag_i, Sv\text{-}ag_i, K\text{-}ag_i, w\text{-}ag_i, CoA_i> ; i = 1,\ldots, N\text{-}a\}$.

Here, *N-a* is the total number of services that can be provided.

$Sv\text{-}ag_i$ contains sva_i, which is determined as a provided function $func_i$ and the provided quality of the $func_i$ $func\text{-}q_i$:

$Sv\text{-}ag_i = \{sva_i \mid sva_i = <func_i, func\text{-}q_i>; i = 1,\ldots, N\text{-}a\}$.

AL-AG contains ag_j (j = 1, .., *N-alt*),

$AL\text{-}AG = \{ag_j \mid ag_j = <na_j, M\text{-}ag_j, Sv\text{-}ag_j, K\text{-}ag_j, nil, CoA_j> ; j=1,\ldots, N\text{-}alt\}$.

Here, *N-alt* is the total number of alternative agents.

STR contains rel_{ij} (i = 1, .., *N-a*, j = 1,…, *N-alt*), and rel_{ij} is determined by r_{ij}, which is the relation between ag_i and ag_j. Here, ag_j are contained in *AG* and r_{ij} are contained in a set of relations between agents *REL*.

$STR = \{rel_{ij} \mid rel_{ij} = <r_{ij}, ag_i, ag_j>; ag_i, ag_j \in AG; rel_{ij} \in REL\}$.

[Definition 4] *F* is a set of functions that behave so as to satisfy the required services in the EAS. They are determined on the basis of a set of $Sv\text{-}ag_i$,

$F = \cup\, Sv\text{-}ag_i, ag_i \in AG : i=1,..., N\text{-}a, N\text{-}n = |F|, N\text{-}r \leq N\text{-}n$. □

When an *R* is issued to the EAS, an *F* is determined in a design process.

[Definition 5] In a design process, the EAS is feasible and an *F* is determined when all requirements and functions are compared and matched,

if $req_k = <req\text{-}f_k, req\text{-}f\text{-}q_k> \in R$ is issued and $sva_i = <func_i, func\text{-}q_i> \in S$ exists,
$req\text{-}f_k = func_i$, and $req\text{-}f\text{-}q_k = func\text{-}q_i$,
then *F* is determined. □

The design process selects those agents that have a function needed to realize the requirement. The agent is organized when an *F* is determined, that is, when *AG*, *STR*, and *AL-AG*, which are elements of *S*, are determined.

[Definition 6] *E* represents a set of information about the behavioral environment. It consists of information about agent platforms *Ag-Plfms*, distributed platforms *Ds-Plfms*, and agent repositories *Ag-Rpgs*:

$E = <$ *Ag-Plfms, Ds-Plfms, Ag-Rpgs* $>$. □

Ag-Plfms contains ap_i (i = 1,…, *N-ap*), where an ap_i is determined by an identifier of the agent platform nap_i, the type of agent platform $ap\text{-}typ_i$, the processing efficiency $ap\text{-}pef_i$, the number of agent processing tasks $Mg\text{-}ag_i$, and a number of functions $S\text{-}AP_i$:

$Ag\text{-}Plaf=\{ap_i \mid ap_i=<nap_i, ap\text{-}typ_i, ap\text{-}pef_i, Mg\text{-}ag_i, S\text{-}AP >; i = 1,\ldots, N\text{-}ap\}$.

Here, *N-ap* is the total number of the agent platforms.

Ds-Plfms contains dp_i (i = 1,…, *N-dp*), where a dp_i is determined as an identifier of a distributed platform ndp_i, the type of distributed platform $dp\text{-}typ_i$, the processing efficiency $dp\text{-}pef_i$, a number of services $Sv\text{-}dp_i$, and a number of functions $S\text{-}AP_i$:

$$Ds\text{-}Plaf=\{dp_i \mid dp_i=<ndp_i, dp\text{-}typ_i, dp\text{-}pef_i, Sv\text{-}dp_i>;\ i=1,\ldots, N\text{-}dp\}.$$

Here, *N-dp* is the total number of distributed agent platforms.

Ag-Plfms consists of the main agent repository *p-AgR* and a set of agent sub-repositories *s-AgRs*:

$$Ag\text{-}Rpgs = <p\text{-}AgR, s\text{-}AgRs>$$

P is the operating property of the EAS. It is determined by the evolutional mechanism of the EAS. If the system can recover its performance, then the evolutional mechanism actively controls the system based on the operating status of the multiagent system and information about the system environment. In this study, we design the evolutional mechanism as a meta-agent *mag*. It manages *P* as internal information.

[Definition 7] *mag* consists of an identifier of *mag* $n\text{-}mag_i$, a set of the *mag*'s received messages $M\text{-}mag_i$, a set of services that can be provided by mag_i $Sv\text{-}mag_i$, a set of *mag*'s action knowledge $K\text{-}mag_i$, the main agent on agent organization *t-ag*, the operation mode *op-mode*, the history of *P*, and the operation property *Prop*:

$$mag = <n\text{-}mag_i, M\text{-}mag_i, Sv\text{-}mag_i, K\text{-}mag_i, t\text{-}ag, op\text{-}mode, History, Prop>$$

Here, *Prop* includes detectable properties $prop_i$ (i = 1, 2, …, *N-p*)

$$Prop = \{prop_i \mid i = 1,\ldots,N\text{-}p\}$$

where *N-p* is the total number of detectable properties. □

3.3. Change and control in EAS

In an EAS, a change during operation is detected as a change in a component of the EAS. For example, if ΔS represents the amount of S changes, then the change in S is $S+\Delta S$.

[Definition 8] In an EAS, changes observed in an observable object *obj(t)* at time *t* describe *Ch(obj(t), ψ, t))*. Here, the change from $t-\Delta t$ to t is represented by *ch(obj(t–Δt), obj(t))* or $\Delta obj(t)$, its rate is represented by *rat(Δobj(t), obj(t))*, or *rat_Δobj(t)*, and ψ is a threshold value of *rat_Δobj(t)*. *Ch(obj(t), ψ, t))* can be represented by the following.

if $\Delta obj(t) \neq 0$ and $rat\_\Delta obj(t) > \psi$,
  where $\Delta obj(t) = ch(obj(t-\Delta t), obj(t))$,
    $rat\_\Delta obj(t) = rat(\Delta obj(t), obj(t))$
then $\Delta obj(t) = Ch(obj(t),\psi, t)$. □

[Definition 9] Among $Ch(obj(t), \psi, t)$ in the EAS, degradation is described by $Dg(obj(t), \varepsilon, t)$, which is an undesirable change for the EAS. Here, ε is a threshold value for the degradation:

if $Ch(obj(t), \varepsilon, t)$ is detected
and $\Delta obj(t) < 0$, $rat\_\Delta obj(t) > \varepsilon$,
then $\Delta obj(t) = Dg(obj(t), \varepsilon, t)$. □

[Definition 10] When a change in the EAS is detected, the EAS behaves as follows: (1) it detects the change, (2) it sets an objective, (3) it operates, and (4) it evaluates the result. After processing, the EAS changes to another EAS: EAS'. Representing the conditions of the components of the EAS as EAS/<*ne*, *R*, *S*, *F*, *E*, *P*>, the transition of EAS is represented by

$$EAS/<ne, R, S, F, E, P> \rightarrow EAS'/<ne, R+\Delta R, S+\Delta S, F+\Delta F, E+\Delta E, P+\Delta P> \square$$

Moreover, the operation in the EAS can be classified into Tuning, Reorganization, E-Reorganization, and E-Redesign. In the Tuning phase, the changed P' is recovered by controlling some parameters, and the EAS is returned to the original P. In the Reorganization phase, the EAS is reorganized when it cannot recover in the Tuning phase. Then an EAS^+ resembling the original EAS is constructed. The difference between EAS and EAS^+ is less than a threshold θ. In the E-Reorganization phase, if a more suitable EAS exists as EAS^*, for which P is P^*, then EAS is reorganized to EAS^*. In the E-Redesign phase, the EAS cannot recover the phases described above. The EAS demands its redesign from the EAS designer.

[Definition 11] Classification of operations is as follows.

Tuning: $EAS' \Rightarrow EAS$, $P' \rightarrow P$,
Reorganization: $EAS' \Rightarrow EAS$, $P' \rightarrow P^+$, $\|P^+ - P\| \leq \theta$,
E-Reorganization: $EAS' \Rightarrow EAS^*$, $P' \rightarrow P^*$, $P^* \geq P$,
E-Redesign: EAS^* with $\delta R \Rightarrow EAS^{**}$. □

When changes are detected, an EAS invokes δF and δS, which are counter operations given to an F and S. Fundamentally, the EAS implements δF and δS against changes. During the E-Reorganization and E-Redesign phases, the counter operation given to P δP is invoked. Only in the Re-design phase is the counter operation given to R (δR) necessary. Operations are classified into the following patterns. Here, A⇒B signifies that the EAS changes from A to B.

1. A change in R: ΔR
   $\Delta R \Rightarrow \delta F, \delta S$
   $EAS'/<ne, R+\Delta R, S, F, E, P'> \Rightarrow EAS^+/<ne, R+\Delta R, S, F+\delta F, E, P^+>|$
   $EAS^{**}/<ne, R+\Delta R, S+\delta S, F+\delta F, E, P^{**}>$
2. A change in S: ΔS
   $\Delta S \Rightarrow \delta F, \delta S$
   $EAS'/<ne, R, S, F+\Delta F, E, P'> \Rightarrow EAS^+/<ne, R, S+\Delta S+\delta S, F+\delta F, E, P^+>|$
   $EAS^*/< ne, R, S+\Delta S+\delta S, F+\delta F, E, P^*>|$
   $EAS^{**}/< ne, R+\delta R, S+\Delta S+\delta S, F+\delta F, E, P^{**}>$
3. A change in F: ΔF
   $\Delta F \Rightarrow \delta F, \delta S$
   $EAS'/< ne, R, S, F+\Delta F, E, P'> \Rightarrow EAS/<ne, R, S, F+\Delta F+\delta F, E, P>|$
   $EAS^+/<ne, R, S+\delta S, F+\Delta F+\delta F, E, P^+>|$

$EAS^{*}/< ne, R, S+\delta S, F+\Delta F+\delta F, E, P^{*}>|$
$EAS^{**}/< ne, R+\delta R, S+\delta S, F+\Delta F+\delta F, E, P^{**}>$

4. A change in P: ΔP
   $\Delta P \Rightarrow \delta F, \delta S$
   $EAS'/<ne, R, S, F, E, P+\Delta P> \Rightarrow EAS/<ne, R, S, F+\delta F, E, P>|$
   $EAS^{+}/<ne, R, S+\delta S, F+\delta F, E, P^{+}>|$
   $EAS^{*}/< ne, R, S+\delta S, F+\delta F, E, P^{*}>|$
   $EAS^{**}/< ne, R+\delta R, S+\delta S, F+\delta F, E, P^{**}>$
5. A change in E: ΔE
   $\Delta E \Rightarrow \delta F, \delta S$
   $EAS'/<ne, R, S, F, E+\Delta E, P'> \Rightarrow EAS/<ne, R, S, F+\delta F, E+\Delta E, P>|$
   $EAS^{+}/<ne, R, S+\delta S, F+\delta F, E+\Delta E, P^{+}>|$
   $EAS^{*}/< ne, R, S+\delta S, F+\delta F, E+\Delta E, P^{*}>|$
   $EAS^{**}/< ne, R+\delta R, S+\delta S, F+\delta F, E+\Delta E, P^{**}>$ □

Finally, we describe the general forms of the EAS. They are summarized functions based on designs and changes.

[Definition 12] General forms of counter operations as yielded by function are shown below.

1. State translations to EAS/EAS^{+}
   Tuning:
   $\delta F \leftarrow TF_i(t, Dg(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$
   Reorganization:
   $\delta S \leftarrow RO_i(t, Dg(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$
   Recover:
   $AS \leftarrow Rcv_i(t, Dg(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$
2. State translations to EAS^{*}/EAS^{**}
   E-Tuning:
   $\delta P \leftarrow EG_i(t, Ch(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$
   $\delta F \leftarrow ET_i(t, Ch(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$
   E-Reorganization:
   $\delta P \leftarrow EG_i(t, Ch(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$
   $\delta S \leftarrow ER_i(t, Ch(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$
   E-Redesign:
   $\delta R \leftarrow$ User: given by user,
   $\delta P \leftarrow EG_i(t, Ch(obj, \varepsilon, t), R, S, F, E, P, \tau_i)$ □

The operations presented above are designed and implemented as the knowledge of a meta-agent. Autonomous operations are achieved because the operational characteristics and the counter operations are implemented in the meta-agent on the basis of detectable changes and conditions of each element in the EAS.

3.4. System margin

To design a control mechanism based on the system margin, we define a margin of tuning *Tm* and a margin of alternative *Al* as the elements of $prop_i$. They are derived by an *R*, *F* and *S*. Based on the formula, *Tm* and *Al* are derived by the EAS at regular intervals during service provision. Observing the changes of *Tm* and *Al*, the EAS evaluates the operational status and controls them.

We define *Tm* as the degree of recovery and stability of QoS by tuning. We quantify the tunable degree of the system based on *Tm* and estimate the "control possibility" as a possibility of tuning. Therefore, *Tm* is derived on the basis of QoS provided and the maximum QoS that can be provided.

[Definition 13] The provided QoS is derived from an *F* in the EAS. A set of services provided to users corresponds to a set of functions that provide services. The set of these functions is *F*. Consolidating *func*-q_is determines the overall QoS that is provided. Here, a set of functions that provides the maximum QoS by tuning is represented by F^*, and *Tm* is derived on the basis of *F* and F^*,

$$Tm = F_{Tm}(F^*, F). \square$$

We define *Al* as the degree of recovery and stability of QoS by reconstruction. We quantify the provided QoS in reconstruction based on *Al* and estimate the "control effect" as an effect of reconstruction. Therefore, *Al* is derived on the basis of the QoS provided and on the QoS that will be provided after reconstruction.

[Definition 14] The QoS provided by a reconstruction is calculated on the basis of *AL_AG*. By reconstructing the EAS, an *AL_AG* is selected as a new member of the EAS, replacing *AG* with *AL_AG*. Here, a set of functions provided by reconstruction is represented as F_{alt}, and *Al* is derived on the basis of *F* and F_{alt}:

$$Al = F_{Al}(F_{alt}, F). \square$$

3.5. Agent behavior design based on *Tm* and *Al*

The architecture of the service-provisioning system based on *Tm* and *Al* is presented in Figure 1 [17]. In the figure, "Agent Workplace" is the behavioral platform that controls the agent organization and each agent to provide services for users. "Agent Repository" is the agent server; it constructs agent organizations and manages the stored agents. "Ag" is an agent providing services to users as the member of the agent organization and is a component of an *AG* and an *AL_AG*. Moreover, "UA," "EA," "META," and "Manager" are used. A UA is a user agent that receives user requirements and reports those requirements to a META. An EA is an environment-monitoring agent that collects platform and network environment information and then reports that information to the META. The META is a meta-agent. Based on information from the UA and the EAs in the Agent Workplace, the META evaluates operating conditions and manages control. A Manager is a manager agent that constructs and manages agent organizations in the Agent Repository.

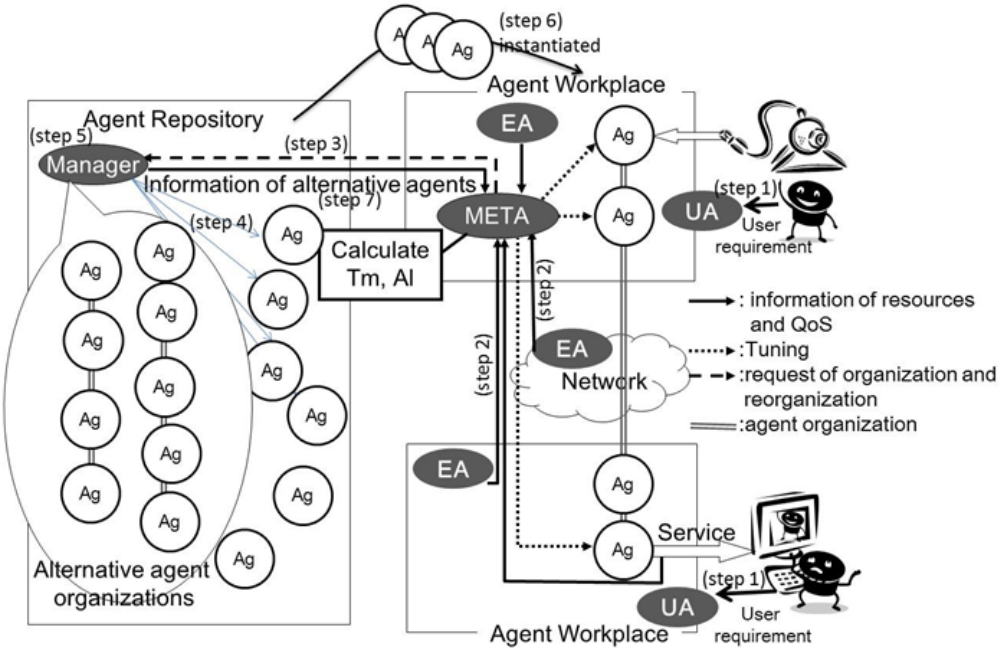

Figure 1. Architecture of the service-provisioning system based on *Tm* and *Al*

Using this architecture, agent behaviors are designed as follows. A UA obtains a requirement from users, and then, the UA informs a META about the requirement. The META requests environment information from EAs. The META sends a service requirement to a Manager on the basis of the user requirement and the platform and network environment information. Then, the Manager constructs an agent organization that satisfies the requirement. The agents instantiate to Agent Workplaces. During service provision, the Manager sends information of *AL_AG*, as an alternative agent, to the META. Then, the META calculates *Tm* and *Al* and controls the EAS. The META garners agent conditions and environment information, predicts the maximum QoS, and calculates *Tm*. Then, *Al* is calculated on the basis of alternative agent information and the QoS to be provided by the agents.

The flow of information of C from A to B is written as C:A → B; the process described above is as follows. The (step1) – (step12) in Figure 1 denotes the following flow of (step1) – (step12).

Flow of information:

(step 1) *R*:User → UA, then R:UA → META,
(step 2) *E*:EA → META,
(step 3) *R*, *E*:META → Manager with request-organization,
(step 4) *R*, *E*:Manager → Ag,
(step 5) *F* is designed by Manager based on *R* and E, *AG* is determined according to *F*,
(step 6) *AG* is instantiated to agent workplace and
(step 7) *AL_AG*:Manager → META.

Calculation of *Tm* and *AL* in META:
(step 8) acquire and update the information of *F* and *E*,
(step 9) select F^* from *Sv*-ag_i in *AG*,
(step 10) design F_{alt} based on *R* and *Sv*-ag_i in *AL_AG*,
(step 11) calculate *Tm* based on *F* and F^* and
(step 12) calculate *Al* based on *F* and F_{alt}.

If *Tm* was degraded and QoS could not be recovered using Tunings and if *Al* was raised and QoS could be recovered by reorganization, then the EAS decides to conduct E-Reorganization. Here, we define thresholds for *Tm* and *Al* as ε_{Tm} and ε_{Al}, respectively, and define thresholds for time durations of *Tm* and *Al* as γ_{Tm} and γ_{Al}, respectively. Reorganization based on the system margin is described by the following.

[Definition 15]
Degradation of system quality:
if $\Delta Tm < 0$ and $rat\_\Delta Tm > \varepsilon_{Tm,}$
then degradation of *Tm* is detected at time *t*, $\Delta Tm = Dg(Tm, \varepsilon_{Tm}, t)$;
if $\Delta Al < 0$ and $rat\_\Delta Al > \varepsilon_{Al,}$
then degradation of *Al* is detected at time *t*, $\Delta Al = Dg(Al, \varepsilon_{Al}, t)$.
Reorganization:
if $\xi_{Tm} \geq \gamma_{Tm}$ and $0 < \xi_{Al} < \gamma_{Al}$
where ξ_{Tm} is the duration of $Dg(Tm, \varepsilon_{Tm}, t)$
and ξ_{Al} is the duration time of $Dg(Al, \varepsilon_{Al}, t)$,
then EAS'/ $<ne, R, S, F, E, P+\Delta P> \Rightarrow$ EAS*/ $<ne, R, S+\delta S, F, E, P^*>$,
where $\delta S = ER_i(t, Dg(Tm, \varepsilon_{Tm}, t), R, S, F, E, P, \tau_i>$. □

By implementing these models as behavioral knowledge of the META, E-Reorganization is realized.

4. Evaluation

4.1. Application to a multimedia communication system

We redesigned and reimplemented a Flexible Multimedia Network Middleware (FMNM) [17, 18] to test the EAS and the system margin. The FMNN is an agent-based multimedia communication system dynamically providing adequate multimedia communication services on the basis of user requirements. Every agent of FMNM is implemented as an ADIPS/DASH agent [19–24]. In this study, we realized not only passive self-recovery of the FMNM against a QoS change but also a forestalling of QoS degradation. Thereby, better service could be provided.

In the FMNM, the quality of the provided service and the platform and network resource conditions are used as QoS parameters in the multimedia communication system. Consequently, in the prototype system of this study, every QoS parameter is implemented as a component of design specifications; the parameters of a user requirement function and its quality are implemented as $req\text{-}f_k$ and $req\text{-}f\text{-}q_k$, respectively. Therefore, if we implement the priority of $req\text{-}f_k$ as $req\text{-}f\text{-}p_k$, *R* is described as follows:

$$R = \{req_k \mid req_k = <req\text{-}f_k, req\text{-}f\text{-}q_k\ req\text{-}f\text{-}p_k > ;\ k=1, .., N\text{-}r\ \}.$$

Every agent has knowledge of the service that the agent can provide via components of a set of *Sv-AG*.

When a user requirement, *R*, is issued, an *F* that fulfills *R*, i.e. F^*, is selected and agent organization, which is required by the FMNM, is performed in the Agent Repository. If there are other agents that can become a member of the *F*, then they are *Sv-AG*. If there are other agent organizations that are not selected as the *F*, then they are alternative agents *Sv-AL-AG*. Consequently, *Tm* is calculated on the basis of *Sv-AG* and *F* and *Al* is calculated using *Sv-AL-AG* and *F*.

4.2. Purpose and environment of experiments

To evaluate the proposed method, we performed an experiment using a prototype multimedia communication system. This experiment demonstrated that the proposed method could provide adequate service; that is, user requirements were satisfied.

We used the degree of satisfaction of user requirements as the measure of the QoS provided by the system. The degree of satisfaction of a user requirement, which is normalized as requirement satisfaction rate rsr_k, is 100% when the user requirement matches the providing QoS parameter of the multimedia communication service. In addition, the degree of satisfaction of the manager requirement, which is normalized as rsr_k, is 100% when the manager requirement matches the unused resource status of the machine and network. rsr_k is derived as follows:

$$rsr_k = (func_k / req\text{-}f\text{-}q_k) * 100 \ [\%].$$

Therefore, considering the priority α_k, the average of rsr_k RSR is described by

$$\mathrm{RSR}=\{\textstyle\sum_k(\alpha_k \times rsr_k)\}/ \sum_k \alpha_k \ [\%](k=1,\ldots,6).$$

We used *Tm* and *Al* as the measures of tuning and reconstructing by the system. Here, if we represent F^* as follows:

$$F^* = \{sva^* \mid sva^* = <func^*_k, func^*\text{-}q_k>;\ k = 1,\ldots, N\text{-}r\} ,$$

considering the priority α_k, *Tm* is derived as

$$Tm = F_{Tm}(F^*, F) = \{\textstyle\sum_k(\alpha_k \times func^*\text{-}q_k) - \sum_k(\alpha_k \times func\text{-}q_k)\}/ \sum_k \alpha_k \ (k=1,\ldots,6).$$

Moreover, if we represent F_{alt} as follows:

$$F_{alt} = \{sva_{alt} \mid sva_{alt} = <func_{alt\,k}, func_{alt}\text{-}q_k>;\ k = 1,\ldots, N\text{-}r\} ,$$

considering the priority α_k, *Al* is derived as

$$Al = F_{Al}(F_{alt}, F) = \{\textstyle\sum_k(\alpha_k \times func_{alt}\text{-}q_k) - \sum_k(\alpha_k \times func\text{-}q_k)\}/ \sum_k \alpha_k \ (k=1,\ldots,6).$$

In this experiment, the service is accredited as adequate when RSR is greater than 90%. When the RSR continues to be 90% or less, some failures occur in the service. In this paper, we present the results of experiments using these four systems:

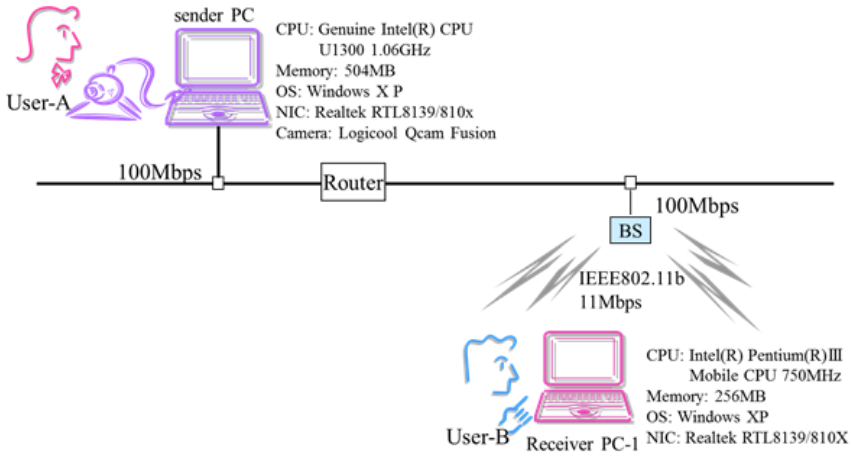

Figure 2. Experimental system configuration

System 1. Conventional multimedia communication system (no agents),
System 2. FMNM using *Tm*,
System 3. FMNM using *Tm* and a reorganization mechanism without *Al*, and
System 4. FMNM using *Tm* and *Al*.

We compared and evaluated the degree of satisfaction of RSR produced by these systems.

Figure 2 depicts the configuration of the experimental system. A sender-side PC was connected to a 100-Mbps Ethernet LAN. A receiver-side PC was also connected to the LAN with an 11-Mbps link of IEEE802.11b via a wireless access network. As the index of QoS, we specifically examined the frame rate of a playing movie as it streamed from the sender PC to the receiver PC. The initial experimental conditions were the following.

$$R = \{req_k \mid req_k = <req\text{-}f_k, req\text{-}f\text{-}q_k, req\text{-}f\text{-}p_k>;\ k = 1,\ldots,\ N\text{-}r\}$$
$$= \{(\text{quality}, 0.6, 0.5), (\text{size}, 58800, 0.5), (\text{frame rate}, 15, 1.5),$$
$$(\text{sender CPU}, 40, 0.8), (\text{receiver CPU}, 40, 0.8), (\text{bandwidth}, 150, 0.8)\}.$$

The threshold of *Tm* was 0.10 and that of *Al* was 0.15.

4.3. Results of experiments

Results of experiments using systems 1–4 are shown in Figures 3–6. In each figure, the x-axis shows time and the y-axis shows the RSR value.

Experimental results for system 1.

Figure 3 presents the experimental results for system 1, which was a conventional multimedia communication system without agent control functions. We used Java Media Framework as the conventional multimedia communication system [25].

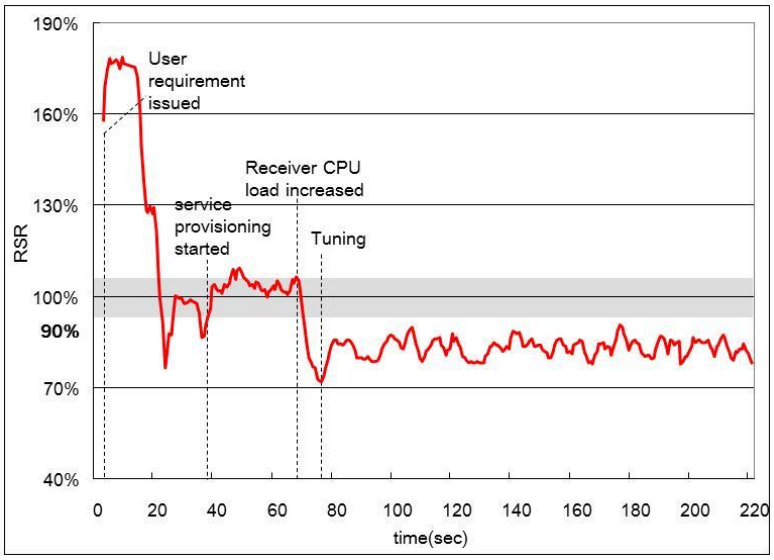

Figure 3. Experimental results for system 1: a conventional multimedia communication system

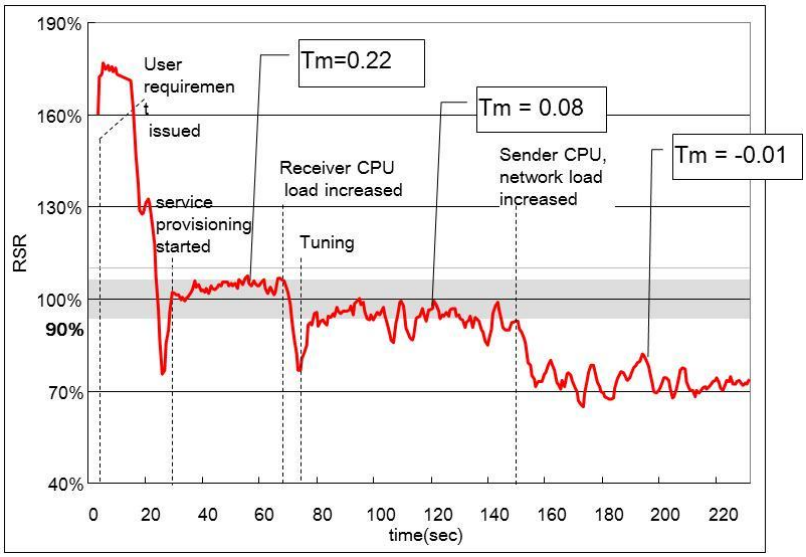

Figure 4. Experimental results for system 2: FMNM based on *Tm*

After service provision started, RSR was more than 90% and QoS was stable; the user requirement was sufficiently satisfied. However, when the CPU load on the receiver PC was increased, RSR decreased. Thereafter, the user requirement was not satisfied because system 1 was incapable of independently controlling the situation.

Experimental results for system 2.

Figure 4 shows the results for service provision with system 2, which was based on an FMNM controlling itself using a *Tm*.

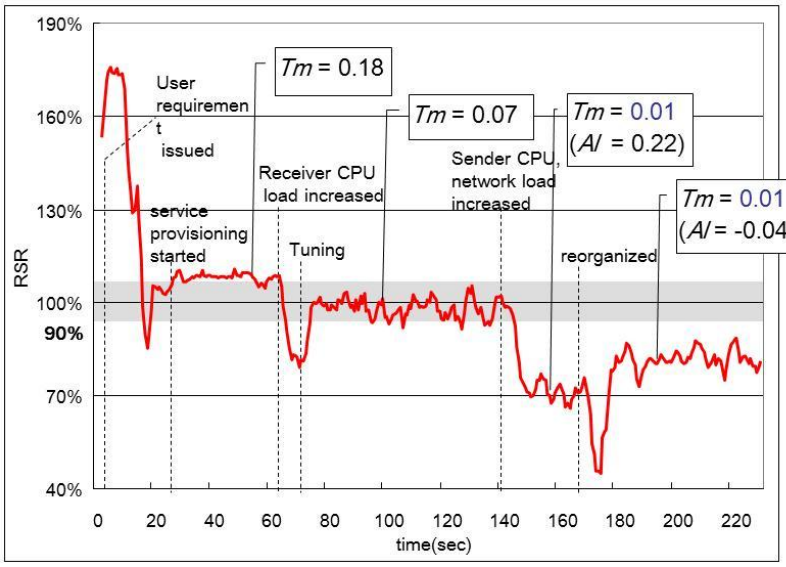

Figure 5. Experimental results for system 3: FMNM based on *Tm* and the conventional reorganization mechanism

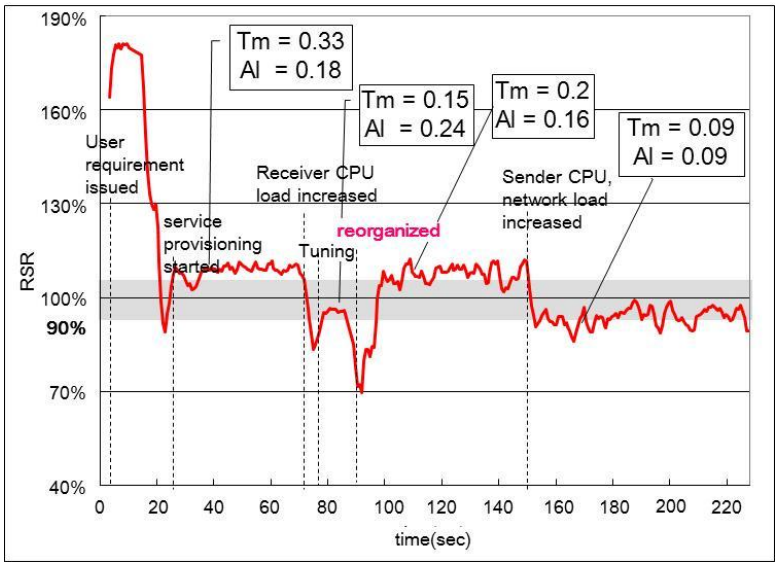

Figure 6. Experimental results for system 4: FMNM based on *Tm* and *Al*

After service provision started, RSR was more than 90% and QoS was stable; the user requirement was satisfied sufficiently. Furthermore, because the value of *Tm* was 0.22, the system was tunable. When the CPU load on the receiver PC was increased, RSR decreased, but the system QoS was recovered through tuning. But as a result, the value of *Tm* decreased to 0.08. Then, when the CPU load on the sender PC was increased, RSR decreased, and the *Tm* decreased to −0.01. At that point, RSR could not be recovered because the system could not be tuned further and because system 2 was incapable of system reorganization.

Experimental results for system 3.

Figure 5 shows the experimental results for system 3. This system was based on an FMNM and controlled it using *Tm* and the FMNM's original reorganization mechanism without *Al*.

After service provision started, RSR was more than 90% and the QoS remained stable; user requirements were satisfied sufficiently. Because the value of *Tm* was 0.18, the system was tunable. When the CPU load on the receiver PC was increased, RSR decreased, but the system QoS recovered through tuning. At that point, the *Tm* value had decreased to 0.07. Then, after the CPU load on the sender PC was increased, RSR again decreased. Now, the *Tm* decreased to 0.01, so the system could not be tuned further. Therefore, the system invoked reorganization. However, because the reorganization mechanism did not consider the effects of reorganization, RSR did not recover.

Experimental results for system 4.

The results in Figure 6 are for system 4. This system was based on an FMNM and controlled it using *Tm* and *Al*.

After service provision started, RSR was more than 90% and the QoS was stable; therefore, the user requirement was satisfied sufficiently. Because the value of *Tm* was 0.33 and that of *Al* was 0.18, the system could be tuned and reorganized. Therefore, when the CPU load on the receiver PC was increased, RSR decreased. But the system QoS recovered through tuning. At that point, the value of *Tm* had decreased to 0.15 and *Al* had increased to 0.24. But because the value of *Tm* was decreasing and the value of *Al* was high and because the system might be stabilized by reorganization, the system conducted E-Reorganization. The results in Figure 6 show that *Tm* recovered to 0.2 and the system was stable again. Consequently, after the CPU load of the sender PC was increased, RSR decreased, but the system remained stable because this system not only could provide service but also could improve the subsequent status of the system.

These results demonstrate that the proposed method maintains stable service against service degradation. Moreover, the proposed method resolves the causes of degradation by estimating the subsequent status of the system, conducting evolutional tuning, and conducting evolutional reorganization.

Using the proposed *Tm*, control incorporating a "control possibility" is achieved when a change of environment occurs. If tuning of QoS is difficult, then the system detects it as a degradation of *Tm* and reconstructs the system. Therefore, a solution to problem (P1) is available.

In addition, using the proposed *Al*, control incorporating a "control effect" is achieved through reorganization of the service provision agents. Moreover, by selecting alternative agent organizations when an agent organization is constructed, reorganization of the agent organization is achieved based on information of an alternative agent organization during service provision. Therefore, a solution to problem (P2) is available.

5. Conclusion

We have proposed a new method for proactively controlling multiagent systems by using "control possibility" and "control effect" to reduce the QoS decrement to the absolute minimum in a service-provisioning system based on multiagency. To evaluate

the effectiveness of the proposed method, we implemented the method in a multimedia communication system. We confirmed that the system can maintain adequate multimedia communication services, and we compared its behavior to those obtained using a conventional multimedia communication system.

In the future, we will apply the proposed method to other types of services, providing systems based on multiagent systems in a ubiquitous environment. Using such systems, we will further evaluate the proposed method and improve the EAS and system margin.

Acknowledgments

This research was partly supported by a Grant-in-Aid for Young Scientists (B), Japan Society for the Promotion of Science, 22700087, 2010-2012, and by a Grant-in-Aid for Scientific Research (C), Japan Society for the Promotion of Science, 22500116, 2010-2012.

References

[1] IBM, http://www.research.ibm.com/autonomic/manifesto/.
[2] G. D. M. Serugendo, M. -P. Gleizes and A. Karageorgos, Infomatica, vol. 30, no. 1, **(2006)**, pp. 45.
[3] A. Takahashi, T. Suganuma, T. Abe, Y. Iwaya and T. Kinoshita, "A Behavioral Characteristics Model for Flexible Distributed System", Proceedings of The IEEE 20th Int. Conf. on Advanced Information Network and Applications (AINA2006), vol. 1, **(2006)**, pp. 275-280.
[4] N. R. Jennings, Artif. Intell., vol. 177, no. 2, **(2000)**, pp. 277.
[5] A. Takahashi, T. Suganuma and T. Kinoshita, IPSJ Journal, vol. 45, no. 2, **(2004)**, pp. 366.
[6] W. H. Oyenan and S. A. DeLoach, "Design and Evaluation of a Multiagent Autonomic Information System", Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, **(2007)**, pp. 182-188.
[7] M. Randles, D. Lamb and A. Taleb-Bendiab, "Engineering Autonomic Systems Self-organization", Proceedings of the Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems, **(2008)**, pp. 107-118.
[8] S. A. DeLoach, W. H. Oyenann and E. T. Matson, Auton. Agent. Multi-ag., vol. 16, no. 1, **(2008)**, pp. 13.
[9] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart and S. R. White, "A Multi-Agent Systems Approach to Autonomic Computing", Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 1, **(2004)**, pp. 464-471.
[10] F. Zambonelli, N. R. Jennings and M. Wooldridge, LNCS, vol. 1957, **(2001)**, pp. 235.
[11] J. M. Serrano and S. Ossowski, WIAS, vol. 5, no. 2, **(2007)**, pp. 197.
[12] W. Jiao, J. Debenham and B. Henderson-Sellers, WIAS, vol. 3, no. 2, **(2005)**, pp. 67.
[13] L. K. Wickramasinghe and L. D. Alahakoon, WIAS, vol. 1, no. 3, **(2005)**, pp. 31.
[14] M. Xu, L. Padgham, A. Mbala and J. Harland, WIAS, vol. 1, no. 5, **(2007)**, pp. 31.
[15] N. Venkatasubramanian, C. Talcott and G. A. Agha, ACM T. on Softw. Eng. Meth., vol. 13, no. 1, **(2004)**, pp. 86.
[16] Y. Liu, A. Fekete and L. Gorton, IEEE Trans. Softw. Eng., vol. 31, no. 11, **(2005)**, pp. 928.
[17] A. Takahashi and T. Kinoshita, IJPCC, vol. 6, no. 2, **(2010)**, pp. 192.
[18] A. Takahashi and T. Kinoshita, WIAS, vol. 9, no. 2, **(2011)**, pp. 161.
[19] S. Fujita, H. Hara, K. Sugawara, T. Kinoshita and N. Shiratori, The International Journal of Applied Intelligence, Neural Network and Complex Problem-Solving Technologies, vol. 9, no. 1, **(1998)**, pp. 57.
[20] T. Kinoshita and K. Sugawara, "ADIPS framework for flexible distributed systems", In Proceedings of Multiagent Platforms, T. Ishida, ed., LNAI, vol. 1599, **(1998)**, pp. 18-32.
[21] T. Uchiya, T. Maemura, L. Xiaolu and T. Kinoshita, "Design and Implementation of Interactive Design Environment of Agent System", Proceedings of 20th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, LNAI, vol. 4570, **(2007)**, pp. 1088-1097.
[22] DASH, "Dash-distributed agent system based on hybrid architecture!", http://www.agent-town.com/dash/index.html.
[23] IDEA, "Idea-interactive design environment for agent system", http://www.k.riec.tohoku.ac.jp/idea/index.html.
[24] T. Uchiya, Y. Nakashima, I. Takumi, T. Kinoshita, H. Hara and K. Sugawara, IJEIC, vol. 2, no. 4, **(2011)**, pp. 47.
[25] JMF, http://java.sun.com/javase/technologies/desktop/media/jmf/.

Appendix

The summary of the definitions of all the notations the figure in section 3 is as follows:

[Definition 1]

ne	The name of the EAS.
R	A set of requirements given to the EAS.
S	The structure of the EAS.
F	A set of functions working in the EAS.
E	The environment in which the EAS operates.
P	The EAS property.

[Definition 2]

req_k	The description of $req\text{-}f_k$ and $req\text{-}f\text{-}q_k$.
$req\text{-}f_k$	A required service function.
$req\text{-}f\text{-}q_k$	The required quality of $req\text{-}f_k$.
N-r	The total number of required functions.

[Definition 3]

AG	A set of member agents of the EAS.
STR	The structure of an agent organization.
AL-AG	A set of alternative agents that can be replaced with a member of *AG*.
ag_i	A member of *AG*.
na_i	The identifier of ag_i.
$M\text{-}ag_i$	A set of received messages of ag_i.
$Sv\text{-}ag_i$	A set of services that can be provided by ag_i.
$K\text{-}ag_i$,	A set of action knowledge of ag_i.
$w\text{-}ag_i$	The processing load of ag_i.
CoA_i	The communication agent set of ag_i.
N-a	The total number of services that can be provided.
sva_i	A member of $Sv\text{-}ag_i$.
N-alt	The total number of alternative agents.
REL	A set of relations between agents.

[Definition 6]

Ag-Plfms	Information about agent platforms.
Ds-Plfms	Information about distributed platforms.
Ag-Rpgs	Information about agent repositories.
ap_i	A member of *Ag-Plfms*.
nap_i	The identifier of the agent platform.
$ap\text{-}typ_i$	The type of agent platform.
$ap\text{-}pef_i$	The processing efficiency.
$Mg\text{-}ag_i$	The number of agent processing tasks of ap_i.
$S\text{-}AP_i$	A number of functions.
N-ap	The total number of the agent platforms.
dp_i	A member of *Ds-Plfms*.
ndp_i	An identifier of a distributed platform.

$dp\text{-}typ_i$	The type of distributed platform.
$dp\text{-}pef_i$	The processing efficiency of dp_i.
$Sv\text{-}dp_i$	A number of services of dp_i.
$S\text{-}AP_i$	A number of functions of dp_i.
N-dp	The total number of distributed agent platforms.
p-AgR	The main agent repository.
s-AgRs	A set of agent sub-repositories.

[Definition 7]

mag	A Meta-agent.
$n\text{-}mag_i$	An identifier of *mag*.
$M\text{-}mag_i$	A set of the *mag*'s received messages.
$Sv\text{-}mag_i$	A set of services that can be provided by mag_i.
$K\text{-}mag_i$	A set of *mag*'s action knowledge.
t-ag	The main agent on agent organization.
op-mode	The operation mode of *mag*.
History	The history of *P*.
Prop	The operation property of *mag*.
N-p	The total number of detectable properties.

[Definition 8]

obj(t)	An observable object.
Ch(obj(t), ψ, t))	A changes observed in an *obj(t)* at time *t*.
ch(obj(t–Δt), obj(t)), Δobj(t)	The change from $t-\Delta t$ to *t*.
rat(Δobj(t), obj(t)), rat_Δobj(t)	A *Δobj(t)*'s rate.
ψ	A threshold value of *rat_Δobj(t)*.

[Definition 9]

Dg(obj(t), ε, t)	A degradation of *Ch(obj(t), ψ, t)*.
ε	A threshold value for the degradation.

[Definition 11]

δF	The counter operation given to an *F*.
δS	The counter operation given to a *S*.
δP	the counter operation given to *P*.
δR	The counter operation given to *R*.

[Definition 13]

Tm	The degree of recovery and stability of QoS by tuning.
F^*	A set of functions that provides the maximum QoS by tuning.

[Definition 14]

Al	The degree of recovery and stability of QoS by reconstruction.
F_{alt}	A set of functions provided by reconstruction.

[Figure 1]

Agent Workplace	The behavioral platform that controls the agent organization and each agent to provide services for users.
Agent Repository	The agent server; it constructs agent organizations and manages the stored agents.
Ag	An agent providing services to users as the member of the agent organization.
UA	A user agent that receives user requirements and reports those requirements to a META.
EA	An environment-monitoring agent that collects platform and network environment information and then reports that information to the META.
META	A meta-agent.

Authors

Akiko Takahashi

She received both a B.E degree in Information Engineering and a M.S degree in Information Science from Tohoku University in 2002 and 2004, respectively, and a Ph.D. degree from Tohoku University in 2007. She is currently an Assistant Professor at Sendai National College of Technology. Her current research interests are intelligent agents, multiagent systems, and ubiquitous computing.

Tetsuo Kinoshita

He received a B.E. degree in electronic engineering from Ibaraki University in 1977, and M.E. and Dr. Eng. degrees in information engineering from Tohoku University in 1979 and 1993, respectively. He is currently a Professor at Research Institute Electrical Communication of Tohoku University. His research interests include knowledge engineering, agent engineering, knowledge-based systems, and agent-based systems. He received the IPSJ Research Award, the IPSJ Best Paper Award, and the IEICE Achievement Award in 1989, 1997, and 2001, respectively. He is a member of IEEE, ACM, AAAI, IEICE, IPSJ, JSAI.