2010 IEEE International Conference on Services Computing

# Dynamic Service Invocation Control in Service Composition Environments

Donghui Lin, Yohei Murakami, Masahiro Tanaka
*National Institute of Information and Communications Technology (NICT),*
*3-5 Hikaridai, Seika-Cho, Soraku-Gun, Kyoto-Fu, 619-0289, Japan*
*Email: {lindh, yohei, mtnk}@nict.go.jp*

***Abstract*—Service composition environments enable people to create, manage, share services, and compose atomic services for their own requirements. Since users and service entity hosts are always distributed in locations in such environments, service responses might be very slow if users invoke services that are physically far from them, which is even slower for invoking composite services. However, those problems cannot be solved using traditional caching technologies in the areas of contents delivery network because service providers in service composition environments always have their own policies about service license issues, and do not allow their service entities to be copied on all service entity hosts. In this paper, we propose the approach of dynamically switching availability of services among service entity hosts based on the invocation request from users, and design several dynamic invocation control mechanisms to improve the response performance of services considering the service license constraints by service providers. Our proposed mechanisms are evaluated by simulation in service composition environments.**

***Keywords*-service composition environment; service invocation control; response performance; service license constraint; composite service**

## I. Introduction

In recent years, with the development of modern Internet and distributed environments, service composition environments have become more and more important, where various kinds of services are gathered within a certain domain or across domains. Such environments enable people to create, manage their own services, and share their services with each other, while users can get additional value of services by composing them based on their own requirements. The Language Grid [1] is a typical example of service composition environments in the domain of language services.

While service composition platforms provide easy access to services, there are several major problems in such environments. First, since physical locations of users and service entity hosts are distributed in different areas around the world, invocation response of services might be very slow if users invoke services that are physically far from the users. Moreover, when invoking a composite service that consists of several atomic services, the invocation response is always much slower if all those atomic services are located in service entity hosts that are far from each other.

Similar problems have also been reported in the areas of contents delivery network (CDN), which can be solved by caching technologies and some other technologies [2]. With such technologies, contents in local servers are fully or partially copied in caching servers around the world to provide quick response for users that are physically distributed. However, it is difficult to apply the same approaches in the areas of CDN to the service composition environments. Since various service providers in service composition environments have their own policies about intellectual property and license issues, service entities are always not allowed to be copied on all service entity hosts. For example, if the service provider allows two licenses of a service to be used in the environments, it should be ensured that two service entities at maximum are available to be invoked at a same time. Therefore, dealing with the invocation response performance problem with license constraint of services is a challenging issue in service composition environments.

To address the above issues, this research focuses on designing dynamic service invocation control mechanisms in service composition environments to improve the response performance of services considering service license constraints by service providers. To achieve this objective, we propose the approach of dynamically switching availability of services among service entity hosts based on the invocation request from users. We first describe the service invocation problem in service composition environments. Then, we propose a series of mechanisms for dynamic service invocation control for both atomic services and composite services. For atomic services, we propose several mechanisms including the request-based invocation control, the user-based invocation control and the hybrid invocation control that combines the above two mechanisms. For composite services, we propose the individual invocation control mechanism and the group invocation control mechanism. The proposed mechanisms are expected to be evaluated based on simulations in service composition platforms in aspects of service response performance and stability of service response.

The rest of the paper is organized as follows: Section II provides the service invocation control problem in service composition environments based on a motivation example. In Section III, we describe a series of dynamic service invocation control mechanisms for both atomic services and composite services. Section IV introduces the evaluation and analysis of proposed mechanisms with experiments. Section V introduces some related work, followed by the conclusion

978-0-7695-4126-6/10 $26.00 © 2010 IEEE
DOI 10.1109/SCC.2010.65

in Section VI.

## II. Service Invocation Problem

### A. Motivation Example

In this section, we first use an example of the Language Grid[1] to show the service invocation problem. The Language Grid provides a service composition environment for users to share, create and combine Web services in language domains, which we call language services [1] . As shown in Figure 1, the infrastructure of Language Grid is aimed at connecting two kinds of servers (core nodes and service nodes). Core nodes manage all requests to language services, while service nodes (or we call service entity hosts) actually invoke atomic services. If the requested service is a composite one as shown in Figure 1, core nodes invoke a corresponding Web service workflow that includes multiple atomic services (Service 1, 2, 3, 6, 7 in Figure 1). Registered information of language services is shared among all core nodes. Users can access the nearest core node to invoke services. Currently, 118 organizations from 17 countries have become users of the Language Grid, and over 50 atomic language services are provided on the Language Grid[2]. The atomic language services can be combined by composite language services.

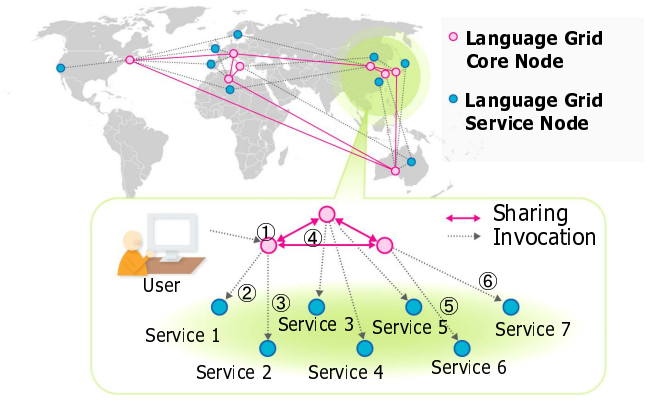


Figure 1. Example of service composition environment: Language Grid

In the Language Grid, we have been facing problems of service invocation. Table I shows some data of response time for service invocation by Language Grid user from Thailand. The user invokes an atomic service (J-Server Japanese-English translation service) and a composite service (composite Japanese-English translation service combined with dictionary which is composed by three atomic services on the Language Grid: J-Server Japanese-English translation service, Mecab Japanese morphological analysis service and the Kyoto tourism Japanese-English dictionary). Each service is invoked for 10 times from both the service entity host in Thailand that is near to the user and the service entity host in Japan that is far from the user. From the result in Table I, we can see that it takes 2.54 times and 4.87 times of response time for invoking an atomic service and a composite service respectively on the host in Japan than in Thailand for Thai user. Similar experiments have been conducted between Language Grid users in Japan and in Denmark. When invoking an atomic service (J-Server Japanese-English translation service) on the service entity host in Japan, user from Denmark costs 5.39 times of response time than user from Japan. Therefore, it is necessary to consider how to improve the performance of service invocation response time in service composition environments.

Table I
Invocation response time of atomic services and composite services on different service hosts by a same user.

| Invocation | Atomic service | | Composite service | |
|---|---|---|---|---|
| | Thailand | Japan | Thailand | Japan |
| 1 | 200ms | 578ms | 285ms | 1474ms |
| 2 | 126ms | 413ms | 290ms | 1429ms |
| 3 | 179ms | 421ms | 310ms | 1437ms |
| 4 | 119ms | 408ms | 293ms | 1446ms |
| 5 | 184ms | 405ms | 250ms | 1438ms |
| 6 | 184ms | 410ms | 326ms | 1452ms |
| 7 | 129ms | 417ms | 264ms | 1444ms |
| 8 | 191ms | 411ms | 411ms | 1486ms |
| 9 | 186ms | 411ms | 253ms | 1440ms |
| 10 | 188ms | 409ms | 293ms | 1434ms |
| *Average* | 168.6ms | 428.3ms | 297.5ms | 1448ms |

Since service providers in service composition environments have their own policies about intellectual property and license issues, service entities are always not allowed to be copied on all service entity hosts. Therefore, dealing with the invocation response performance problem with license constraint of services is an important issue in service composition environments.

### B. Service Invocation in Service Composition Environments

In the service composition environments, there are multiple users and service entity hosts that are distributed in different locations. The environments enable users to invoke atomic services, or composite services with each consisting of a group of atomic services. Each atomic service can have limited number of service entities. Each service entity can be deployed at any service entity host. Service invocation response time depends on the location of user and service entity host. The objectives of service invocation control in such environments include the following three aspects: (1) Invocation response performance for atomic services should be high; (2) Invocation response performance for composite services should be high; and (3) Invocation response performance should keep stable for all users within the environments.

[1] http://langrid.nict.go.jp/

[2] Lists of Language Grid users and services are available in the Language Grid Service Manager (http://langrid.org/service_manager/)

Table II
NOTATIONS IN SERVICE INVOCATION PROBLEM.

| Notation | Description |
|---|---|
| $u_i$ | user $i$ $(i = 1, 2, \ldots, m)$ |
| $s_i$ | atomic service $i$ $(i = 1, 2, \ldots, n)$ |
| $t_i$ | invocation time period $i$ $(i = 1, 2, \ldots, p)$ |
| $h_i$ | service entity host $i$ $(i = 1, 2, \ldots, l)$ |
| $H(s_i)$ | set of service entity hosts where $s_i$ is invocable |
| $r_i(u_j, s_k)$ | invocation request $i$ $(i = 1, 2, \ldots, v)$ (to atomic service $s_j$ from user $u_i$) |
| $c(u_i, s_j, h_k)$ | invocation response time for $s_j$ on $h_k$ from $u_i$ |
| $w_i$ | composite service $i$ $(i = 1, 2, \ldots, z)$ |
| $S(w_i)$ | set of atomic services for composite service $w_i$ |
| $r_i(u_i, w_j)$ | invocation request $i$ $(i = 1, 2, \ldots, w)$ (to composite service $w_j$ from user $u_i$) |
| $c(u_i, w_j)$ | invocation response time for $w_j$ from $u_i$ |

Table II shows the notations and their descriptions used in the service invocation problem in the service composition environments. Each service invocation request is a request to a certain service $s$ by a certain user $u$. Therefore, we use $r(u, s)$ to denote a service invocation request. For each atomic service $s$, there might be several service entities and each exists in a certain service entity host $h$. We use $H(s) = \{h_1(s), h_2(s), \ldots, h_q(s)\}(q = |H(s)|)$ to denote the set of current service hosts for service $s$ where service entity of $s$ are deployed and can be invoked. $q$ is the maximum license number available for service $s$. Each composite service $w$ is composed by a group of atomic services $S(w) = \{s_1, s_2, \ldots, s_k\}$ with some control flows between them, e.g., sequential relation, parallel relation, selective relation and so on. Therefore, invocation request to an atomic service $s$ might be a request of $s$ itself or a request of composite service $w$ where $s$ is among the atomic services that consist $w$. Basically, composite service can be regarded as a service-based workflow, which has the general characteristics of workflow. Therefore, many theorems and systems of workflow management area can be applied. For example, workflow patterns [3] proposed in previous research can also be used to design composite services. Invocation response time of a composite service $w$ can be gained from the aggregation of invocation response time of the atomic services that consist it. In previous research, aggregation of QoS functions for composite service has been proposed [4]. In this research, we use the aggregation approach based on workflow patterns [5].

## III. DYNAMIC SERVICE INVOCATION CONTROL MECHANISMS

To address the issues of service response problems in service composition environments where both Web service entities and users are located distributedly, we propose several dynamic service invocation control mechanisms in this section. The objective is to improve the performance of service invocation request under constraints of policies of service providers and physical locations. The following information can be acquired in the service composition environments: (1) nearest service host to user $u$ with fastest response; (2) current nearest service host to user $u$ where service $s$ can be invoked; (3) atomic services that consist a composite service $w$; (4) history request count for invoking atomic service $s$ or composite service $w$ by user $u$ at time period $t$; and (5) request reservation information for invoking atomic service $s$ or composite service $w$ by user $u$ at time period $t$.

### A. Invocation Control for Atomic Service

*1) Request-based Invocation Control:* The most straight and simple approach for invocation control of atomic service $s$ is to dynamically make the service entity of $s$ available for invocation on the nearest service host to the user $u$ for each request $r(u, s)$, which is called request-based invocation control. With the limitation of maximum possible license number of service entities, it is necessary to control the availability of service entity among service hosts, which can be realized by service entity migration [6] [7] among service hosts or license control using the framework like floating license [8]. Every time when a request for service invocation comes, the requested service entity is always controlled to be available on the nearest service entity host to the user as shown in **Algorithm 1**. However, the biggest problem of the request-based invocation control is that there might be many times of changing service host of service $s$ by service migration or license control and therefore it always costs much time to control the availability of service entity among service host.

**Algorithm 1** Request-based invocation control mechanism for atomic service

**procedure** AtomicServiceInvocation1 ($s$)
1: **for all** service request $r_i(u, s)$ **do**
2: $h^* \leftarrow nearestServiceHost(u)$
3: **if** $h^* \notin H(s)$ **then**
4: $h^s \leftarrow currentNearestServiceHost(u, s)$
5: Switch availability of $s$ from service host $h^s$ to $h^*$
6: $H(s) \leftarrow (H(s) \setminus \{h^s\}) \cup \{h^*\}$
7: **end if**
8: **end for**

*2) User-based Invocation Control:* Invocation control of switching availability of service entities among service hosts always costs time. Therefore, we propose the user-based invocation control mechanism to reduce the switching times among service hosts, where availability of service entities are controlled periodically based on the service invocation trends from users. Users are always from different areas in service composition environments such as the Language Grid. We have observed the invocation requests from users vary in different time periods. Based on the history invocation request information and invocation request reservation information from users, we can anticipate the service invocation request

trend from users in different time periods. Therefore, the user-based invocation control mechanism is designed to minimize the times of switching availability of service entity among service hosts. As is described in **Algorithm 2**, based on the information of possible invocation request $ut[u_i][s]$ of users for each time period, service entities of a service $s$ is always dynamically controlled to be available on the service hosts which are nearest to the users $U^*(s)$ with the most possible invocation requests of $s$. Switching of service entity availability among service hosts always occur at the beginning of each time period, while there is no switching of service entity availability during the time period. However, although minimum cost of switching of service entity is required in user-based invocation control, it might lack of flexibility to deal with the cases when real service invocation request situation is not the same as what is anticipated.

**Algorithm 2** User-based invocation control mechanism for atomic service

**procedure** AtomicServiceInvocation2 ($s$, $t$)
1: **for all** user $u_i$ **do**
2: $ut[u_i][s] \leftarrow historyRequest(u_i, s, t)$
3: **if** ($hasReservedRequest(u_i, s, t) = 1$) **then**
4: $ut[u_i][s] \leftarrow reservedRequest(u_i, s, t)$
5: **end if**
6: **end for**
7: Sort $ut[u_i][s]$ for all users
8: $U^*(s) \leftarrow \phi$
9: Add users with the $|H(s)|$ maximum $ut[u_i][s]$ to $U^*(s)$ ($|H(s)|$ is service entity number of $s$)
10: **for all** $u_i \in U^*(s)$ **do**
11: $h^* \leftarrow nearestServiceHost(u_i)$
12: **if** ($h^* \notin H(s)$) **then**
13: $h^s \leftarrow currentNearestServiceHost(u_i, s)$
14: Switch availability of $s$ from service host $h^s$ to $h^*$
15: $H(s) \leftarrow (H(s) \setminus \{h^s\}) \cup \{h^*\}$
16: **end if**
17: **end for**
18: **for all** service request $r_j(u, s)$ **do**
19: Update $historyRequest(u, s, t)$
20: **end for**

*3) Hybrid Invocation Control:* As is mentioned above, the request-based invocation control mechanism has the problems of high cost due to frequent change of service host, while the user-based invocation control mechanism lacks of flexibility to deal with the real service invocation request situation when it is different with the anticipated situation. To consider both the switching cost of service hosts and flexibility of handling different service invocation request situations in real cases, we propose the hybrid invocation control mechanism by combining the above two approaches for different cases, which is shown in **Algorithm 3**. For each time period, service entity of service $s$ is initially controlled based on the user-based invocation control mechanism where service entities are switched to be available on the service hosts that are nearest to the users with the most potential requests. During the time period, service invocation requests are monitored in real time. If situations become different from what is anticipated, the available service entities will be dynamically switched to the nearest service host to the user with continuous service invocation requests of service $s$ which exceeds the control bound $cr(s)$.

**Algorithm 3** Hybrid invocation control mechanism for atomic service

**procedure** AtomicServiceInvocation3 ($s$, $t$)
1: AtomicServiceInvocation2 ($s$, $t$)
2: $H^0(s) \leftarrow H(s)$
3: $cr^* \leftarrow 1$
4: Set $cr(s)$ (control bound of continuous request for $s$ from a same user)
5: **for all** service request $r_i(u, s)$ **do**
6: **if** ($i > 1$) **then**
7: **if** ($user(r_i(u, s)) = user(r_{i-1}(u, s))$) **then**
8: $cr^* \leftarrow cr^* + 1$
9: **else**
10: $cr^* \leftarrow 1$
11: **end if**
12: **end if**
13: **if** $cr^* > cr(s)$ **then**
14: $h^* \leftarrow nearestServiceHost(u)$
15: **if** $h^* \notin H(s)$ **then**
16: $h^s \leftarrow currentNearestServiceHost(u, s)$
17: Switch availability of $s$ from service host $h^s$ to $h^*$
18: $H(s) \leftarrow H(s) \cup \{h^*\} - \{h^s\}$
19: **end if**
20: **else if** $u \in U^*(s)$ **then**
21: Switch availability of $s$ so that current set of service entity host returns to $H^0(s)$
22: **end if**
23: **end for**

### *B. Invocation Control for Composite Services*

*1) Individual Invocation Control:* In service composition environments, many atomic services are provided. What is more important is that such environment provides users additional values of various new services by composing atomic services. Composite services consist of multiple atomic services and are always used more frequently than atomic services. Simple invocation control mechanism for composite service $w$ can be realized by applying invocation control mechanisms for all individual atomic services $S(w) = \{s_1, s_2, \ldots, s_n\}$ that consist the composite service $w$. The average response time for invoking composite service $w$ can be expected to be reduced by applying

invocation control mechanism for each individual atomic service $s_i(i = 1, 2, \ldots, n)$. However, since the individual invocation control mechanism does not consider the group characteristics of atomic services, applying optimal invocation control mechanisms for atomic services individually does not guarantee that it is also optimal for the whole composite service.

*2) Group Invocation Control:* Since each composite service consists of a group of atomic services, we propose the group invocation control mechanism for composite service based on the group characteristics of atomic services. That is, the group invocation control considers the group of atomic services that consist composite service together rather than separately. For example, if atomic service $s_a$, atomic service $s_b$ and atomic service $s_c$ have high possibilities to be used together in a composite service by users, then these three services can be considered as a service group when deciding the invocation control policies in the service composition environment.

The request-based invocation control mechanism for composite service is shown as **Algorithm 4**. For each composite service invocation request $r(u, w)$, the request-based invocation control mechanism dynamically makes the service entities of all atomic service $s_i(i = 1, 2, \ldots, n)$ that consist the composite service available for invocation on the nearest service host to the user $u$ by migration among service hosts or license control of service entities.

**Algorithm 4** Request-based invocation control mechanism for composite service

**procedure** CompositeServiceInvocation1 ($w$, $t$)
1: $S(w) \leftarrow servicesInWorkflow(w)$
2: **for all** service $s_i \in S(w)$ **do**
3: AtomicServiceInvocation1 ($s_i$)
4: **end for**

The user-based invocation control mechanism for composite service is shown as **Algorithm 5**, which is designed to minimize the times of switching availability of service entity among service hosts for all atomic services that consist the composite service. Based on the information of possible composite service invocation request $ut[u_i][w]$ of users for each time period, service entities of all atomic service $s_i(i = 1, 2, \ldots, n)$ are always controlled to be available on the service hosts which are nearest to the users $U^*(w)$ with the most possible invocation requests of $w$.

The hybrid invocation control mechanism for composite service is shown as **Algorithm 6**. For each time period, service entities of all atomic service $s_i(i = 1, 2, \ldots, n)$ is initially controlled based on the user-based invocation control mechanism for composite service. During the time period, if situations become different from what is anticipated, the available service entities for each atomic service will be dynamically switched to the nearest service host to the user with continuous service invocation requests, which is similar to the hybrid invocation control mechanism for atomic service.

**Algorithm 5** User-based invocation control mechanism for composite service

**procedure** CompositeServiceInvocation2 ($w$, $t$)
1: $S(w) \leftarrow servicesInWorkflow(w)$
2: **for all** user $u_i$ **do**
3: $ut[u_i][w] \leftarrow historyRequest(u_i, w, t)$
4: **if** ($hasReservedRequest(u_i, w, t) = 1$) **then**
5: $ut[u_i][w] \leftarrow reservedRequest(u_i, w, t)$
6: **end if**
7: **end for**
8: Sort $ut[u_i][w]$ for all users
9: **for all** service $s_j \in S(w)$ **do**
10: $U^*(s_j) \leftarrow \phi$
11: Add users with the $|H(s_j)|$ maximum $ut[u_i][w]$ to $U^*(s_j)$ ($|H(s_j)|$ is service entity number of $s_j$)
12: Switch availability of $s_j$ to nearest service hosts to all users in $U^*(s_j)$ (the same as line 10-17 in Algorithm 2)
13: **end for**
14: **for all** service workflow request $r_k(u, w)$ **do**
15: Update $historyRequest(u, w, t)$
16: **end for**

**Algorithm 6** Hybrid invocation control mechanism for composite service

**procedure** CompositeServiceInvocation3 ($w$, $t$)
1: $S(w) \leftarrow servicesInWorkflow(w)$
2: CompositeServiceInvocation2 ($w$, $t$)
3: **for all** service $s_i \in S(w)$ **do**
4: Switch availability of $s_i$ to nearest service host to the current user that have continuous invocation requests over $cr(s_i)$ (the same as line 2-23 in Algorithm 3)
5: **end for**

## IV. Experiments and Analysis

### A. Experiments

*1) Settings:* We simulate the service composition environments by setting the user number $n = 100$ and the service entity host number $m = 20$. For each user $u_i$, the invocation response time of atomic service $s$ on any service entity host $c(u_i, s, h_j)$ (ms) is randomly simulated with different probabilities as follows which is similar with the response time distribution on the Language Grid: (1) $p(100 \leq c(u_i, s, h_j) < 200) = 0.1$; (2) $p(200 \leq c(u_i, s, h_j) < 500) = 0.2$; (3) $p(500 \leq c(u_i, s, h_j) < 1000) = 0.2$; (4) $p(1000 \leq c(u_i, s, h_j) < 2000) = 0.4$; (5) $p(2000 \leq c(u_i, s, h_j) < 5000) = 0.1$. For each atomic

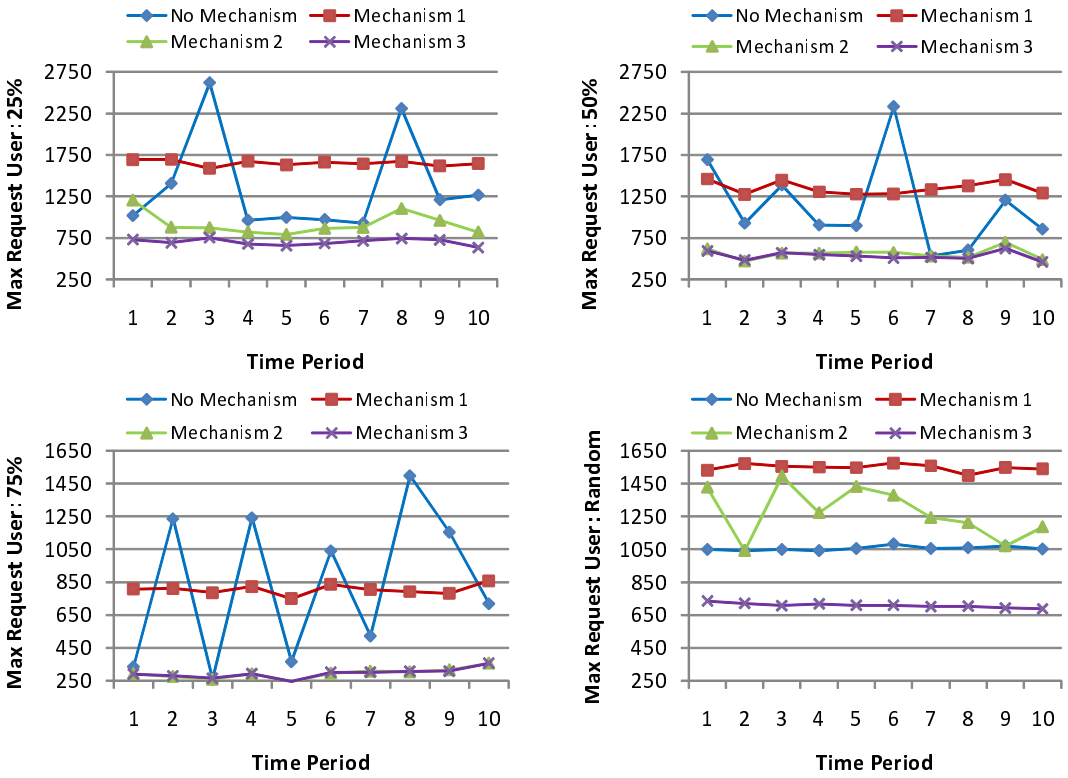


Figure 2. Simulation result of response performance of atomic services

service, the time cost for switch availability from one service entity host to another host is randomly decided between 1000ms and 5000ms since the time cost varies according to the approaches (migration of service entity or license control) of switching availability. During the experiments, we simulate 10 time periods for each service and invoke the service for 5000 times in each time period of simulation. Moreover, for each invocation request to an atomic service or a composite service, we set 50% as the probability that current request is a continuous request from the same user. For each service $s_i$, the invocation requests are totally random in sequence of users. For each time period, $|H(s)|$ users are set to have the most invocation request times based on the history request information and request reservation information (request from the $|H(s)|$ users is set to be 25%, 50%, 75% or a random percentage of the total request from users to service $s$). Each composite service is set to be randomly composed by three to five atomic services with the sequential patterns and parallel patterns. In the simulations, $c(u_i, w)$, the response time for a composite service $w$ from $u_i$ can be aggregated by the invocation response time of atomic services that consist $w$.

*2) Measurement:* Based on the experiment settings, we conduct three measurements: response performance of atomic service, response performance of composite service, and the stability of response performance. Response performance of a service is evaluated by computing the average service invocation response time from all users for ten time periods (10*5000 invocations). To evaluate the response performance of atomic service, we compare the response performance of atomic service for invocation without dynamic mechanism and the three dynamic invocation control mechanisms. To evaluate the response performance of composite service, we compare the response performance of composite service for invocation without dynamic mechanism, the individual invocation mechanism and the group invocation mechanism. Further, we compare the distribution of average response time from all the users for invocation without dynamic mechanism and dynamic invocation mechanisms to evaluate the stability of response performance.

### B. Analysis

We analyze the experiment results in three aspects.

*1) Response Performance of Atomic Services:* From the results shown in Figure 2, we find that Mechanism 1 (request-based invocation control), Mechanism 2 (user-based invocation control), Mechanism 3 (hybrid invocation control) can all bring average stable response for different invocations. Moreover, the hybrid invocation control mechanism that considers both potential users for most service invocation requests and potential users for continuous requests always performs best. Further, the response performance of user-based invocation control mechanism will approach that

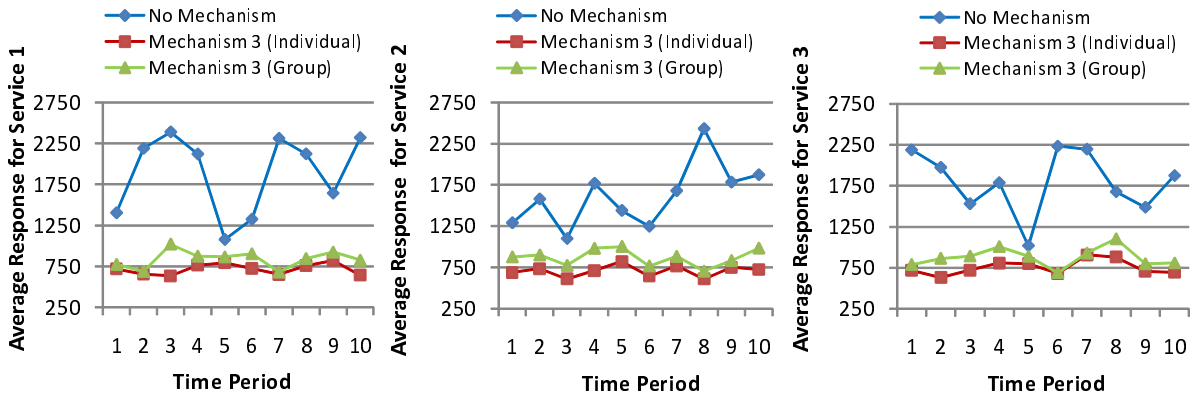


Figure 3. Influences of response performance of composite services on response performance of atomic services

of hybrid invocation control mechanism if the users that mostly request for service invocation are dominant (cover over 50% of the total requests).

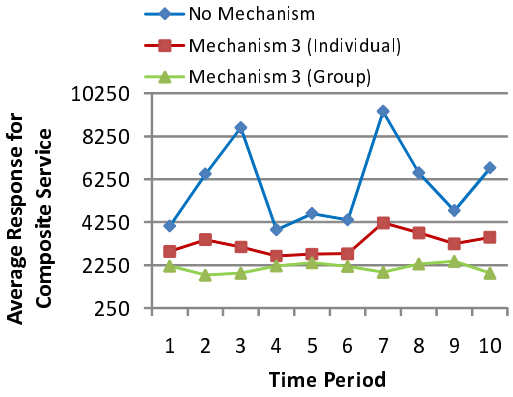


Figure 4. Simulation result of response performance of composite services

*2) Response Performance of Composite Services:* Since hybrid invocation control mechanism for atomic service can bring the best response performance of atomic services, we use it in individual invocation control mechanism and group invocation control mechanism for composite services. In Figure 3, Service 1, Service 2 and Service 3 compose the composite service we use for the experiments with sequential patterns. From the result in Figure 3 and Figure 4, we can see that (1) using group invocation control for composite service can get better response performance comparing to individual invocation control; (2) using group invocation control for composite service may insignificantly decrease the response performance for each atomic service since optimal mechanism for composite service does not guarantee it is also optimal for all its component atomic services.

*3) Stability of Response Performance from the Perspective of Users:* Figure 5 shows a distribution chart of average response time for 100 users to an atomic service by comparing the invocation without dynamic control and the hybrid invocation control mechanism. The result shows that average response time for all users is distributed from 100ms to 5000ms when the dynamic invocation control mechanisms are not applied. However, the average response time for all users becomes stable after using the hybrid invocation control mechanism although it is not necessarily reduced. Even if we vary the number of users or repeat the experiment in different settings, similar phenomena are observed. That means dynamic invocation control mechanism can bring relatively stable response performance from the perspective of users.

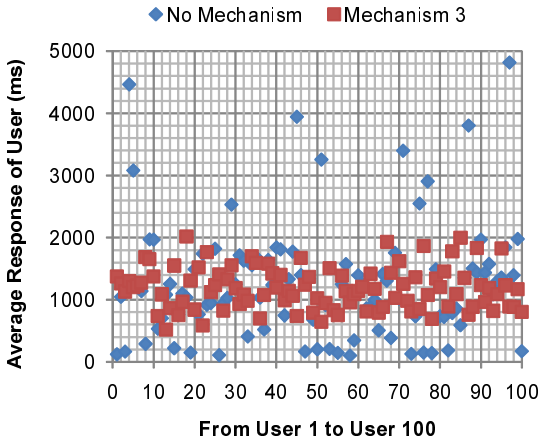


Figure 5. Distribution chart of average response time for 100 users

## V. Related Work

Web service composition has been an important issue for past several years in the service-oriented computing area. Recently, QoS-aware service composition has become the focus in this area. The work of Zeng *et al.* [4] is among the earliest ones for QoS-aware service composition. The authors propose a multidimensional QoS model for Web service composition and several optimization approaches for service selection in both static environment and dynamic environment. Although we do not deal with all the QoS dimensions for Web service composition, we consider the

real problems and constraints in the service composition environments and propose mechanisms to improve response performance for atomic services and composite services.

Dynamic service deployment and invocation has been discussed in previous research in the area of grid computing, where the issue of efficient and reliable resources sharing is always the focus [6] [9]. In service composition environments, services are always wrapped by resources provided by various service providers for different purposes. Therefore, policies of service providers might bring the constraints of service license problems that we deal with in this research, which is different from the situation of the grid computing.

In the areas of contents delivery network (CDN), there are also problems of response performance due to the distribution in physical locations of users and contents servers, which is worked out to be solved by caching technologies [2], which enables contents to be fully or partially copied in caching servers to provide quick response for users. However, service entities are not always allowed to be copied on all service entity hosts in service composition environments since service providers have different policies about service license issues. Therefore it is difficult to apply the caching technology to the service composition environments. Instead, we propose the approach of dynamically switching availability of services among service entity hosts to deal with the service license constraints.

## VI. Conclusion

In service composition environments, service invocation response performance is an important issue since users and service entity hosts are always physically distributed. This paper deals with the issues of service invocation control in service composition environments considering the service license constraints by service providers.

The main contribution of this paper is to propose the approach of dynamically switching availability of services among service entity hosts, and design dynamic invocation control mechanisms for both atomic services and composite service to improve the response performance. For atomic services, we proposed several mechanisms including (1) the request-based invocation control that always makes the requested service available on the nearest service entity host of the current request user, (2) the user-based invocation control that considers the invocation request history and reservation information, and (3) the hybrid invocation control that combines the above two mechanisms. For composite services, we proposed the individual invocation control mechanism based on the respective invocation control of atomic services, and the group invocation control mechanism considering the group characteristics of atomic services that consist the composite service.

The proposed dynamic service invocation control mechanisms were evaluated by simulations in service composition environments. The experiments results shows that (1) For atomic services, the hybrid invocation control mechanism that considers both potential users for most service invocation requests and potential users for continuous requests can improve the response performance best; (2) For composite services, group invocation control can improve the response performance more but may insignificantly decrease the response performance for each atomic service; and (3) Dynamic invocation control mechanisms can bring relatively stable response performance from the perspective of users.

Our future work will be conducted in two aspects. One is to introduce other QoS dimensions of composite services. The other aspect is to propose a user-centered QoS model for the service invocation control problem.

## Acknowledgments

This work was supported by Strategic Information and Communications R&D Promotion Programme (SCOPE) from the Ministry of Internal Affairs and Communications of Japan.

## References

[1] T. Ishida, "Language grid: An infrastructure for intercultural collaboration," in *IEEE/IPSJ Symposium on Applications and the Internet (SAINT-06)*, 2006, pp. 96–100.

[2] M. Pathan and R. Buyya, "A taxonomy of CDNs," *Content Delivery Networks*, pp. 33–78, 2008.

[3] W. Van Der Aalst, A. Ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and parallel databases*, vol. 14, no. 1, pp. 5–51, 2003.

[4] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Transactions on software engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[5] M. Jaeger, G. Rojec-Goldmann, and G. Muhl, "QoS aggregation for Web service composition using workflow patterns," in *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*. IEEE Computer Society Washington, DC, USA, 2004, pp. 149–159.

[6] E. Byun and J. Kim, "DynaGrid: A dynamic service deployment and resource migration framework for WSRF-compliant applications," *Parallel Computing*, vol. 33, no. 4-5, pp. 328–338, 2007.

[7] F. Casati and M. Shan, "Dynamic and adaptive composition of e-services," *Information systems*, vol. 26, no. 3, pp. 143–163, 2001.

[8] N. Bontis and H. Chung, "The evolution of software pricing: from box licenses to application service provider models," *Internet Research: Electronic Networking Applications and Policy*, vol. 10, no. 3, pp. 246–255, 2000.

[9] T. Friese, M. Smith, and B. Freisleben, "Hot service deployment in an ad hoc grid environment," in *Proceedings of the 2nd international conference on Service oriented computing*. ACM, 2004, p. 83.