A Framework for Measuring Performance in Service-Oriented Architecture

Jin Sun Her, Si Won Choi, Sang Hun Oh, and Soo Dong Kim

Department of Computer Science, Soongsil University
511 Sangdo-Dong, Dongjak-Ku, Seoul, Korea 156-743
{jsher, swchoi, shoh}@otlab.ssu.ac.kr, sdkim@ssu.ac.kr

Abstract

Service-Oriented Architecture (SOA) is emerging as a promising approach for delivering applications by composing loosely coupled services. However, SOA has a performance problem due to the loosely coupled and heterogeneous nature of the approach. To practically and widely apply SOA in developing applications, performance problem has to be overcome. For this, it has to be measured how low the performance is and analyzed where and why the problem has occurred. Prerequisite for this is a definition of well-defined metrics for measuring service performance. Well-defined metric here means a metric that is precise and practical enough to effectively diagnose the cause of the performance problem. However, current works on service performance are not precise enough to be applied in effective diagnosis especially regarding the dynamism of SOA. Hence, in this paper, we first define a set of precise and practical metrics for measuring service performance. We then apply the metrics in Hotel Reservation Service and show the applicability and usefulness of the metrics.

1. Introduction

Service-Oriented Architecture (SOA) provides a powerful mechanism for delivering business processes by composing loosely coupled services. Service provider publishes services in service registry, and service client discovers and composes services to carry out the expected service requirement [1]. However, SOA approach has a negative impact on the performance of an application due to physical distribution of services, overhead caused by intermediaries that handle communication, use of a standard message format, etc. [2][3]. Therefore, to achieve the benefits of SOA such as increased organizational agility and reduced implementation cost, performance problem has to be overcome.

To overcome the performance issue, it has to be measured how low the performance is and analyzed where and why the problem has occurred. Prerequisite for this is a definition of well-defined metrics for measuring service performance. Well-defined metric here means a metric that is precise and practical enough to effectively diagnose the cause of the performance problem. Especially dynamism and several design alternatives of SOA have to be considered.

- There are two types of service; atomic service and composite service [4][5].
- Services can be discovered, composed, and adapted either at run time or design time [1][2][6][7].
- Services can be stateless or stateful [1][2][7][8].
- Services can be adaptable or non-adaptable [1][2][6][7].

Performance analysis and cause of the problem may differ in each case. However, current works on service performance are not precise and practical enough to effectively diagnose the cause of the performance problem considering these design alternatives.

Therefore, the goal of this paper is to define a set of precise and practical metrics for measuring service performance. We first briefly investigate the current works on service performance in section 2. Then we define a set of metrics for measuring service performance in section 3. To show the practicability and usefulness of the metrics, we apply the metrics to Hotel Reservation Service in section 4 and give the interpretation in section 5.

2. Related Work

Zhang's work proposes an accountability framework to make service process deployments manageable and dependable in Service-Oriented Computing (SOC) [9].

0-7695-3022-2/07 $25.00 © 2007 IEEE
DOI 10.1109/NWESP.2007.24

The work defines a four-step logic flow of service process deployment. Among the four steps, accountability model is applied in the third step, *service process execution*. In the step, they use Bayesian network to diagnose the root causes of malfunctions and service level agreement (SLA) violations.

Song proposes a performance analysis methodology for web services [10]. In the methodology, they define performance metrics and five activities that combine test- and simulation-based performance analysis methodology to save cost and time. However, this work needs to be extended with the metrics to measure and monitor dynamic composition and adaptation.

Kim's work suggests priority allocation method to improve web-service performance and model for differentiated web-service [11]. They derive priority value for assigning the value to classified messages from response time, satisfaction rate, throughput in the service-level-agreement, accessibility, and reliability for differentiating web-services. Therefore, customer's requests are served according to the priority level.

3. Metrics for Service Performance

3.1. Service Response Time

Service Response Time (SRT) is the elapsed time between the end of a request to a service and the beginning of the service's response. To finish a service transaction, some services only need a single interaction, while some services need multiple interactions. Therefore, in case of a service with single interaction, SRT is the time taken to complete the transaction, while in case of s service with multiple interactions, SRT is the time taken to complete one interaction of the transaction.

To measure service response time, inherent nature of SOA like the following should be considered [2][3].

- Services and service clients are located in different containers, most often on different machines.
- The use of a standard messaging format, XML, increases the time needed to process a request.
- Services may be discovered at either design time or runtime through the use of discovery services or agents.
- Services composing a business process may be adapted to adhere to the business process's need.
- Service composition may need to be adapted by adding additional services or swapping in the adapted services.

The main reason for low performance in SOA usually stems from these factors. Therefore it is significant to define the sub-metrics of service response time considering these factors as in Figure 1. In the figure, BP stands for business process, SRV stands for service, and SS stands for secondary storage.

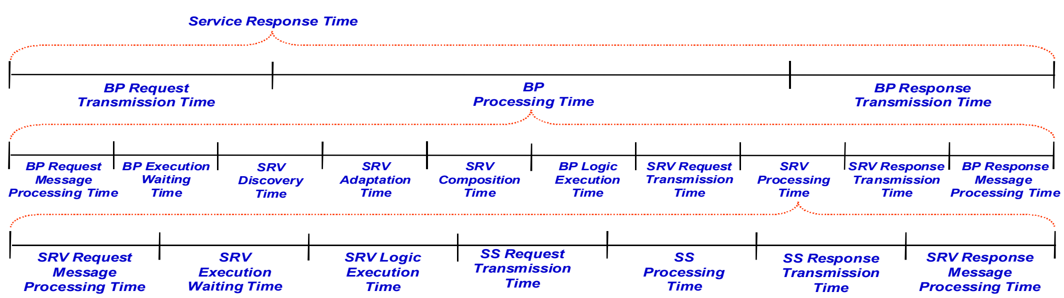

Figure 1. Decomposition view of sub-metrics composing SRT

Based on Figure 1, we propose 20 sub-metrics as summarized in Table 1 in terms of metric name, abbreviation, and formula.

Table 1. Sub-metrics of SRT

Metric		Formula
Name	Abbr.	
BP Request Transmission Time	*TTReq(BP)*	*Time when BP receives request from Service Client – Time when Service Client sends request to BP*
BP Processing Time	*PT(BP)*	*Time when BP sends response to Service Client – Time when BP receives request from Service Client*
BP Response Transmission Time	*TTRes(BP)*	*Time when Service Client receives response from BP – Time when BP sends response to Service Client*
BP Request Message Processing Time	*ReqMsg(BP)*	*Time when processing BP Request Message Finishes – Time when processing BP Request Message Starts*
BP Execution Waiting Time	*WT(BP)*	*Time when BP Request Message Leaves the Queue – Time when BP Request Message Arrives at the Queue*
SRV Discovery Time	*DT(SRV)*	*Time when the required service is discovered – Time when BP starts to find the required service from Service Registry*
SRV Adaptation Time	*AT(SRV)*	*Time when service adaptation is finished – Time when service adaptation is started*
SRV Composition Time	*CT(SRV)*	*Time when service composition is finished – Time when service composition is started*
BP Logic Execution Time	*ET(BP)*	*PT(BP) – (ReqMsg(BP)+ WT(BP) + DT(SRV) + AT(SRV) + CT(SRV) + TTReq(SRV) + PT(SRV) + TTRes(SRV) + ResMsg(BP))*

SRV Request Transmission Time	*TTReq(SRV)*	*Time when Service receives request from BP – Time when BP sends request to Service*
SRV Processing Time	*PT(SRV)*	*Time when Service sends response to BP – Time when Service receives request from BP*
SRV Response Transmission Time	*TTRes(SRV)*	*Time when BP receives response from Service – Time when Service sends response to BP*
BP Response Message Processing Time	*ResMsg(BP)*	*Time when processing BP Response Message Finishes – Time when processing BP Response Message Starts*
SRV Request Message Processing Time	*ReqMsg(SRV)*	*Time when processing SRV Request Message Finishes – Time when processing SRV Request Message Starts*
SRV Execution Waiting Time	*WT(SRV)*	*Time when SRV Request Message Leaves the Queue – Time when SRV Request Message Arrives at the Queue*
SRV Logic Execution Time	*ET(SRV)*	*PT(SRV) – (ReqMsg(SRV) + WT(SRV) + TTReq(SS) + PT(SS) + TTRes(SS) + ResMsg(SRV))*
SS Request Transmission Time	*TTReq(SS)*	*Time when SS receives request from SRV – Time when SRV sends request to SS*
SS Processing Time	*PT(SS)*	*Time when SS sends response to SRV – Time when SS receives request from SRV*
SS Response Transmission Time	*TTRes(SS)*	*Time when SRV receives response from SS – Time when SS sends response to SRV*
SRV Response Message Processing Time	*ResMsg(SRV)*	*Time when processing SRV Response Message Finishes –Time when processing SRV Response Message Starts*

The value range of these metrics is *value* ≥ 0, where higher value of the metric indicates worse performance.

ReqMsg(SRV), ResMsg(SRV), ReqMsg(BP), and *ResMsg(BP)* are for measuring the time spent in parsing, validating, and transforming XML documents. *ET(SRV)* and *ET(BP)* are for measuring the time spent in executing the logic that service and business process implements, thus these metrics always have the value of more than zero.

However, *WT(SRV)* and *WT(BP)* are considered when there are a number of messages waiting in the ready queue. *TTReq(SS)*, *PT(SS)*, and *TTRes(SS)* are considered only when the service is a stateful service that needs to store the states in the secondary storage.

A business process can provide service by the logic implemented in itself without invoking external web services, therefore the value of the metrics, *TTReq(SRV)*, *PT(SRV)* and *TTRes(SRV),* is 0 in this case. *DT(SRV), AT(SRV),* and *CT(SRV)* are considered when services are discovered, adapted, and composed at run time dynamically.

Service Response Time (SRT) is composed of *TTReq(BP), TTRes(BP), and PT(BP)* or *PR(SRV).* An atomic service implemented as a web service and a composite service implemented as a business process are accessible through the WSDL published in the service registry. Therefore, service client may invoke either or both of them as the following formula.

SRT = *Time when Service Client finishes sending request to the service (atomic or composite service) – Time when Service Client starts receiving response from the service (atomic or composite service)*

However the metric compositions described above do not always disjoint. According to the implementation structure of the service, one metric may be a part of another metric.

3.2. Think Time

Think Time (TH) is an elapsed time between the end of a response generated by a service and the beginning of an end user's response. Therefore, think time can be applied to a service with several interactions. The range of this metric is *TH* > 0, where higher value of *TH* denotes lower performance since it increases the turnaround time. *TH* can be applied to a service that requires end user's response.

Total Think Time (TH_{Total}) metric is the total time spent for thinking within one turnaround.

Think Rate (TR) is proportion of the time end user has spent for thinking to make a reaction to the service's response. *TR* can be computed as TH_{Total}/STA. The range of this metric is 1..0, where the higher value indicates that the user's thinking time contributed largely in resulting low performance.

3.3. Service Turnaround Time

Service Turnaround Time (STA) is a time needed to get the result from a group of related activities within a transaction. In many cases one turnaround time includes many service responses.

Let there be *m* number of *Service Response Time (SRT)* within a turnaround and *n* number of *Think Time (TH)* within a turnaround. And let SRT_i be the i^{th} *SRT* and TH_j be the j^{th} *TH*. Then *STA* metric can be computed like the following.

$$STA = \sum_{i=1}^{m} SRT_i + \sum_{j=1}^{n} TH_j$$

That is, a service turnaround time is a summation of all the service response times and think times within one transaction. The range of this metric is *STA* > 0, where higher value of *STA* denotes lower performance. *STA* can be applied to a service with several interactions. In case of a service with just one interaction, *STA* is identical to *SRT*.

3.4. Throughput

Throughput represents the number of requests served at a given period of time. In SOA, throughput metric is defined in terms of evaluation target; service and business process. Therefore, we propose two metrics; *TP(SRV)* for the *throughput of a service* and

TP(BP) for the *throughput of a business process*. They can be calculated with the following formula.

$$TP(SRV) = \frac{Number\ of\ Completed\ Service\ Requests}{Unit\ of\ Time}$$

Here, the numerator is the number of successfully completed requests to a web service exposed by WSDL. And the denominator is unit of time such as second, minute, or hour.

$$TP(BP) = \frac{Number\ of\ Completed\ Business\ Process\ Requests}{Unit\ of\ Time}$$

Here, the numerator is the number of successfully completed requests to a business process implemented by BPEL. The value range of these two metrics is *TP(SRV)* > 0 and *TP(BP)* > 0 respectively. Higher value indicates a better performance. And how many requests can be processed means how many users can be processed concurrently in a web.

3.5. Service Performance

In this section, we show the final metrics of *Service Performance* using the metrics defined in the previous sections. We give metrics for measuring the amount of work processed and the time efficiency, which are namely called *Throughput Ratio* and *Response Time Ratio*. The metrics are defined in Table 2 in terms of metric, formula, value range, and interpretation.

Table 2. Final metrics of service performance

Metric	Formula	Range	Interpretation
Through-put Ratio	*TP / Required TP,* Numerator is the currently measured TP which can be either TP(SRV) or TP(BP). Denominator is the required TP that is defined in the SLA.	≥ 0	Near 1 is meeting the requirement, Higher value is better
Service Response Time Ratio	*Sum of SRT / Required Sum of SRT,* *n*; number of interactions in a service. Numerator is the currently measured summation of the *n* SRTs. Denominator is the required summation of the *n* SRTs that is defined in the SLA.	> 0	Near 1 is meeting the requirement, Lower value is better

4. A Case Study: Total Hotel Reservation

In this section, we apply the metrics in *Total Hotel Reservation Service System (THRSS)* to show the practicability and usefulness of the metrics.

4.1. Service Scenario of THRSS

Evaluation target of the case study is *THRSS* which is developed in Chang's work [12]. Overall workflow of the scenario consists of 10 flows as shown in Figure 2. This scenario focuses on being able to support service composition and adaptation.

(a) Traveler enters a THRSS
(b) The traveler enters ID and PW
(c) The THRSS analyzes the traveler's preference and history based on the traveler's ID
(d) The traveler enters information for reserving hotel such as period, area, room type, and hotel grade
(e) THRSS selects a hotel finding service from candidate hotel finding services, which is competitive to find hotels in the area the traveler enters
(f) The competitive hotel finding service returns a list of candidate hotels and their WSDL information for service interface by which THRSS can access to each hotel
(g) THRSS checks the current availability of the individual hotel
(h) THRSS takes individual information for currently available hotels by using the returned WSDLs
(i) THRSS returns the information
(j) THRSS reserve the hotel based on the traveler's preference

Figure 2. Service scenario of THRSS

4.2. Environment Description

We first briefly explain the mechanism on how *THRSS* is operated and then describe environment for measuring the service [12]. Internal mechanism of THRSS is as following: (a) traveler client sends a request message for reserving a hotel to queue of activeMQ, (b) the message is transferred into ESB, (c) ESB routes the message to the business process, (d) business process executes its activities and invokes external web services through ESB, and (e) the external web service executes the request and return the result to ESB in a similar fashion. In order to apply suggested metrics to *THRSS*, we define components, *StatisticsCollector* and *PerformanceMonitor,* to acquire raw data needed for computing the metrics and to compute metrics based on the raw data. *PerformanceMonitor* is deployed on ESB to calculate suggested metrics because ESB routes all messages between business process and web services. *StaticsCollector* is deployed on business process layer, ESB layer, and web-service layer to gather each raw data.

4.3. Measuring Metrics

In this section, we apply the selected metrics; *AT(SRV), CT(SRV), PT(SRV), SRT, TH,* TH_{Total}*, TR, STA, Service Response Time Ratio*, and *Service Response Time Ratio*, and show the value of the metrics computation.

In order to apply performance related metrics, we first collect time information using *captureCurrentTime( )* in *StatisticsCollector* which is implemented by *System.nanoTime( )*. Table 3 shows collected time information for measuring *SRT, CT(Dynamic Composition Handler), AT(Interface Adapter),* and *PT(User Profile Analyzer)* in Figure 3.

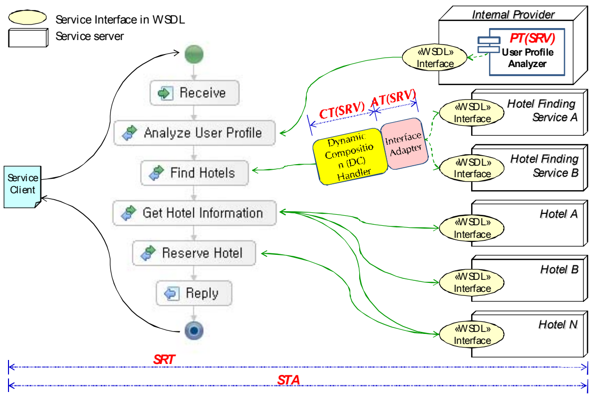

Figure 3. Measured metrics with service architecture

In the table, (a) is time when service client finishes sending request to the service, (b) is time when service client starts receiving response from the service, (c) is time when service composition is finished, (d) is time when service composition is started, (e) is time when service adaptation is finished, (f) is time when service adaptation is started, (g) is time when service sends response to BP, and (h) is time when service receive request from BP. The unit of the time is nanosecond.

Table 3. Collected time information

Service Client	*Dynamic Composition Handler*	*Interface Adapter*	*User Profile Analyzer*
(a) 361286016988059	(c) 361286333231541	(e) 361286399554903	(g) 361286392219209
(b) 361286541588179	(d) 361286410591915	(f) 361286410509067	(h) 361286392409576

In order to measure *Service Response Time Ratio*, two measures are required; (a) time when service client finishes sending request to the service and (b) time when service client starts receiving response from the service. Table 4 captures the result of collected time data for 10 times which will be used in the two measures.

Table 4. Result of collected time data for 10 times

	1st	*2nd*	...	*10th*
(a)	*361286016988059*	*362808891789600*	...	*369797651757532*
(b)	*361286541588179*	*362809419339795*	...	*369797983367395*

Based on these collected data in table 3 and 4, *Performance Monitor* calculates the metrics. *Performance Monitor* component consists of five classes for calculating the metrics; *SRVResTimeCalculator, SRV TurnaroundTimeCalculator, PerformanceCalculator, ThroughputCalculator,* and *ThinkTimeCalculator.* In Table 5, (i) represents methods for computing *AT(SRV)* metric and *CT(SRV)* metric in *SRVResTimeCalculator* class. (ii) represents a method for computing *STA* metric in *SRVTurnaroundTimeCalculator* class. (iii) represents a method for computing *Service Response Time Ratio* metric in *PerformanceCalculator* class.

Table 5. Implementation of performance monitor

(i)	public double computeSRVAdaptation Time(CollectedData data){
	SRVAdaptationTime = data.getTime_Finish_ServiceAdaptation() - data.getTime_ Start_ServiceAdaptation();
	return SRVAdaptationTime;
	}
	public double computeSRVCompo sitionTime (CollectedData data){
	SRVCompositionTime=data.getTime_Finish_ServiceComposition()-data.getTime_Start_Service Composition();
	return SRVCompositionTime;
	}

(ii)	public double computeSRV TurnaroundTime(){
	double totalSRT = 0;
	for (int i = 0; i < SRVResTime CalList.size(); i++){
	SRVResTimeCalculator SRVResT = (SRVResTimeCalculator)SRVResTimeCalList.get(i);
	totalSRT += SRVResT.getSRVResponseTime();
	}
	SRVTurnaroundTime = totalSRT + thinkT.getTotalThinkTime();
	return SRVTurnaroundTime;
	}

(iii)	public double computeSRT_Ratio(){
	double measured_SRT_Ratio = 0;
	if (situation.equals(MeasurementSituation.MeaSit01)){
	measured_SRT_Ratio = SRTCalculator.getSumOfSRT() / pfSLA.getRequiredSumOf SRT();
	}
	else if (situation.equals(MeasurementSituation.MeaSit02)){
	measured_SRT_Ratio = SRTCalculator.getMeanSumOfSRT() / pfSLA.getRequired MeanSumOfSRT();
	}
	return measured_SRT_Ratio;
	}

Table 6 represents the value for each metric which are calculated from table 3 and table 5. In table 6, the value of *TH, TH_{Total}, and TR* is zero because the thinking time does not exist in this case study. The value of *STA* is same as *SRT* because *TH* is zero. The value of *Service Response Time Ratio* is derived from table 4 and *Required Mean Sum of SRT* is from the SLA which is defined as 1 second.

Table 6. Result of each metric

Metrics	Values	Metrics	Values
AT (Interface Adapter)	10954164	TH_{Total}	0
CT (Dynamic Composition Handler)	77360374	*TR*	0
PT(User Profile Analyzer)	190367	*STA*	524600120
SRT	524600120	*Service Response Time Ratio*	0.040264
TH	0		

5. Interpretation of the Metrics

In this section, we show the usefulness of the proposed metrics by clarifying usage of the metrics and interpretation of the metrics.

Usage of the metrics. Service providers can use the metrics to analyze the performance of the services they provide, diagnose where the performance problem has occurred, and tune the performance statically and dynamically. Service providers can make the best use of the metrics since they can collect proprietary data which the other users may not access. Service brokers can use the metrics to provide better search information to the consumers, provide quality service endpoints to the consumer and make their service registry reliable. Service consumers can use the metrics for comparing the services they will select, for discovering services dynamically, and composing services dynamically. In addition, the metrics can be used in defining the details of performance requirement in SLA.

Interpretation of the metrics. Interpretation of the metrics supports the usage of the metrics described above. Due to the paper limitation, we give a brief description of the interpretation in Table 7.

Table 7. Interpretation of the metrics

Metric	Interpretation
Service Response Time	High value may indicate the service design needs optimization or the detailed metrics of the SRT should be analyzed to identify the problem.
Transmission Times	High value of the six transmission times may indicate the network traffic is heavy, or the size of the message is too large, or there is any possibility of reducing communications to a few messages.
BP Processing Time	High value may indicate that the BP process itself needs optimization.
Message Processing Times	High value of the four message processing times may indicate parsing, validating, and transforming the XML documents should be more efficiently processed.
BP & SRV Execution Waiting Time	High value may indicate there are many service clients accessing BP or SRV. If needed, more instances should be prepared.
SRV Discovery Time	High value may indicate that discovery mechanism should be tuned. Typically, dynamic discovery takes more time than static discovery.
SRV Adaptation Time	High value may indicate that adaptation mechanism should be tuned. Typically, dynamic adaptation takes more time then static adaptation.
SRV Composition Time	High value may indicate that composition mechanism should be tuned. Typically, dynamic composition takes more time then static composition.
SRV Processing Time	High value may indicate that the web service itself needs optimization.
SS Processing Time	High value may indicate database tuning is needed.
Think Rate	High value indicates that the user spent high portion of time among the total service execution. This may indicate the overall understandability or usability of the service responses is low.
Service Turnaround Time	High value indicates that either the service system or the service user took most of the time.
Throughput of a Service or BP	Low value may indicate the service or BP needs optimization or the server needs upgrade or more instances are needed.
Throughput Ratio	Less than 1 indicates that optimization should be started.
Service Response Time Ratio	More than 1 indicates that optimization should be started.

6. Conclusion

To overcome the performance issue, it has to be measured how low the performance is and analyzed where and why the problem has occurred. Prerequisite for this is a definition of well-defined metrics for measuring service performance. In this paper, we define a set of precise and practical metrics for measuring service performance. Our metrics are differentiated from the current researches in that they considered the unique activities performed in SOA such as service discovery, adaptation, composition, and invocation. In addition, we considered the programming nature of SOA such as distributed environment and standard messages. To show the practicability and usefulness of the metrics, we applied them in *Hotel Reservation Service*. We demonstrated how we gathered the data by using *StatisticsCollector* and computed the metrics using the *PerformanceMonitor*.

Reference

[1] Erl, T., *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall, 2005.

[2] Brien, L., Bass, L., and Merson, P., *Quality Attributes and Service-Oriented Architectures*, Technical Note CMU/SEI-2005-TN-014, September 2005.

[3] Woodall, P., et al., "Investigating service-oriented system performance: a systematic study," *Software: Practice and Experience*, Vol.37, Issue.2, 2007.

[4] OASIS, *Web Services Business Process Execution Language (BPEL) Version 2.0*, Public Review Draft, 23rd August, 2006.

[5] Arsanjani, A., *Service-Oriented Modeling and Architecture*, IBM Developerworks, 2004.

[6] Marks, E. and Bell, M., *Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology*, Wiley, 2006.

[7] Newcomer, E. and Lomow, G., *Understanding SOA with Web Services*, Addison-Wesley Professional, 2004.

[8] Ferguson, D. and Stockton, M., "Service-Oriented Architecture: Programming Model and Product Architecture," *IBM Systems Journal*, Vol. 44, No. 4, pp. 753-780, 2005.

[9] Zhang, Y., Lin, K., and Hsu, J., "Accountability Monitoring and Reasoning in Service-Oriented Architectures," *The Journal SOCA*, Vol. 1, No. 1, 2007.

[10] Song, H., et al., "Metric, Methodology, and Tool for Performance-Considered Web Service Composition," *In the Proceedings of ISCIS 2005, LNCS 3733*, pp. 392-401, November 2005.

[11] Kim, D., et al., "Improving Web Services Performance using Priority Allocation Method," *In the Proceedings of NWeSP 2005*, pp. 22-26, August 2005.

[12] Chang, S., et al., "Design of a Dynamic Composition Handler for ESB-based Services," *In the Proceedings of ICEBE 2007*, 24-26 October, 2007. (To appear)