Toward Development of Adaptive Service-Based Software Systems

Stephen S. Yau, *Fellow*, *IEEE*, Nong Ye, *Senior Member*, *IEEE*, Hessam S. Sarjoughian, *Member*, *IEEE*, Dazhi Huang, Auttawut Roontiva, Mustafa Gökçe Baydogan, and Mohammed A. Muqsith

Abstract—The rapid adoption of service-oriented architecture (SOA) in many large-scale distributed applications requires the development of adaptive service-based software systems (ASBS) with the capability of monitoring the changing system status, analyzing, and controlling tradeoffs among various quality-of-service (QoS) aspects, and adapting service configurations to satisfy multiple QoS requirements simultaneously. In this paper, our results toward the development of adaptive service-based software systems are presented. The formulation of Activity-State-QoS (ASQ) models and how to use the data from controlled experiments to establish ASQ models for capturing the cause-effect dynamics among service activities, system resource states, and QoS in service-based systems are presented. Then, QoS monitoring modules based on ASQ models and SOA-compliant simulation models are developed to support the validation of the ASBS design. The main idea for developing QoS adaptation modules based on ASQ models is discussed. An experiment based on a voice communication service is used to illustrate our results.

Index Terms—Design concepts, distributed/Internet-based software engineering tools and techniques, methodologies, modeling methodologies, quality of services, services systems.

1 Introduction

Recent development of service-oriented computing and grid computing has led to rapid adoption of service-oriented architecture (SOA) in distributed computing systems, such as enterprise computing infrastructures, grid-enabled applications, and global information systems. Software systems based on SOA are called *service-based software systems (SBS)*. Unique characteristics of SOA, such as loosely coupling and late binding, provide the capabilities that enable the rapid composition of the needed services from various service providers for distributed applications and the runtime adaptation of SBS. However, fundamental changes to existing software engineering techniques are needed for the design of high-quality SBS due to the unique characteristics of SOA. A major problem to achieve this goal is how to manage the satisfaction of multiple quality-of-service (QoS) requirements simultaneously, such as timeliness, throughput, accuracy, security, dependability, survivability, and availability. Because the satisfaction of the requirement of a QoS aspect often affect the satisfaction of other QoS aspects, tradeoffs of requirements among multiple QoS aspects must be taken into consideration in the design of SBS.

Existing software engineering techniques do not support the analysis and adaptive control of such tradeoffs due to lack of comprehensive understanding of such tradeoffs and their relations to service activities in networked computing systems. Furthermore, an SBS often comprises services owned by various providers. Each service may support multiple service configurations, each of which defines the runtime properties of the service, such as authentication mechanism, priority, and maximum bandwidth reserved for the service, and may affect the runtime performance of the service. The services in SBS often operate in highly dynamic environments, where the services may become temporarily unavailable due to various system and network failures, overloads or other causes. Hence, high-quality SBS need to have the capability to monitor the changing system status, analyze and control tradeoffs among multiple QoS aspects, and adapt service configurations accordingly. Such an SBS is referred as an *Adaptive SBS (ASBS)*.

In this paper, we will present our results toward the development of ASBS. After discussing the related work in Section 2 and the overview of our ideas toward developing ASBS in Section 3, we will present in Section 4 our Activity-State-QoS (ASQ) models for capturing the cause-effect dynamics among service activities, system resource states, and QoS in service-based systems, and how the ASQ models are established using data from experiments. Three critical QoS aspects, timeliness, throughput, and accuracy, are considered in our ASQ models due to their importance in many applications. In Section 5, we will show how QoS monitoring modules are developed based on our ASQ models. In Section 6, we will present how SOA-compliant simulation models are developed to support the validation of ASBS design. The main idea for developing QoS adaptation modules based on ASQ models will also be discussed.

• *The authors are with the Information Assurance Center and the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, PO Box 878809, Tempe, AZ 85287-8809.*
E-mail: {yau, nongye, hessam.sarjoughian, dazhi.huang, auttawut.roontiva, mustafa.baydogan, mmuqsith}@asu.edu.

Manuscript received 15 Jan. 2009; revised 2 Apr. 2009; accepted 18 June 2009; published online 29 June 2009.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2009-01-0006. Digital Object Identifier no. 10.1109/TSC.2009.17.

2 Related Work

Currently, software design is usually based on logic-based operational models. However, performance models linking service operations to resource states and QoS performance are needed for QoS monitoring and adaptation (M/A) in ASBS. Individual QoS aspects related to various activities and resource states in computing and communication systems have been studied both empirically and theoretically [1], [2], [3]. However, how system activities and resource states affect the tradeoffs among various QoS aspects has not been sufficiently investigated [4], [5], and the lack of such knowledge makes the design of ASBS difficult.

Recently, more research has focused on the design and adaptation of systems to satisfy various QoS requirements [6], [7], [8], [9], [10]. QoS-aware service composition [6], [7], [8] aims at finding optimal or suboptimal service composition to satisfy various QoS constraints, such as cost and deadline, within a reasonable amount of time. Various techniques have been presented for QoS-aware service composition, such as service routing [6] and genetic algorithms [7], [8]. However, the QoS models considered in existing QoS-aware service composition methods are usually simple, and runtime adaptation of service composition cannot be efficiently handled by most QoS-aware service composition methods. Self-tuning techniques and autonomic middleware providing support for service adaptation and configuration management have been developed in autonomic computing [9], [10]. In [9], an online control approach was presented to automatically minimizing power consumption of CPU while providing satisfactory response time. In [10], an architecture was presented for an autonomic computing environment (AUTONOMIA) with various middleware services for fault detection, fault handling, and performance monitoring. However, these techniques and middleware are not designed for SBS, which is more loosely coupled and more difficult to control.

In order to provide a cost-effective way for validating whether the design of an ASBS with QoS M/A capabilities will meet the QoS requirements for the ASBS, it is necessary to have proper modeling and simulation support for ASBS, which cannot be provided by existing simulation approaches [11], [12], [13], [14], [15]. The simulation environment Discrete Event System Specification (DEVS)/DOC [11], which supports distributed object computing concepts, may be used to design distributed software systems, but it is not suitable for ASBS due to lack of modeling constructs which are needed for describing QoS aspects in SBS. Other simulation approaches and tools, such as HLA [12] and OPNET [13], are not suitable for developing ASBS unless they are extended with new abstractions that can describe key characteristics of service-oriented computing. The UML 2.0 has been suggested to support simulating software systems, but its concept of time and execution protocol is limited [14]. The process specification and modeling language (PSML) [15] is useful in designing, implementing, and testing services with simulation support, but it lacks direct support for simulating time-based services, which is important for describing dynamics of ASBS. Our simulation models are based on service-oriented computing concepts and will play a central role in validating the proposed design approach for ASBS with multiple QoS.

3 Overview of Development of ASBS

The unique characteristics of SOA, including loosely coupled, late binding, dynamic service availability, and the potentially large number of services available to be used in SBS, create the following significant differences between conventional software systems and SBS:

- Services are autonomous entities that can be used by many users and usually operate in environments not fully known and controlled by users and developers. Hence, the performance of SBS is more difficult to predict than conventional software systems.
- SBS often span across the boundaries of multiple organizations, which makes centralized management difficult.
- SBS often have many services providing similar functionality, which gives users the opportunity to select services more suitable for their needs. Furthermore, service discovery and late binding make SBS more adaptable than conventional software systems.

Due to these differences, new techniques for developing high-quality SBS are needed. Fig. 1 shows a conceptual view of our ASBS, in which functional services in the ASBS and the modules for QoS M/A form a closed control loop [16]. The QoS monitoring modules collect the measurements of various QoS aspects as well as relevant system status, which are used by the QoS adaptation modules to adapt service compositions and/or service configurations in ASBS to satisfy various QoS requirements simultaneously. Our ideas toward the development of ASBS mainly focus on the following three aspects:

1. Methodology for service performance modeling to capture the system dynamics affecting various QoS aspects and their tradeoffs by analyzing data collected in controlled experiments.
2. Approaches to developing QoS M/A modules in ASBS based on service performance models and users' QoS requirements in ASBS.
3. SOA-compliant simulation models for validating ASBS with QoS M/A modules.

Since ASBS in various application domains usually have different QoS requirements, resource requirements, and workload patterns, we have selected a group of example application scenarios utilizing various multimedia services for the development and evaluation of our results.

The scalability of our ASBS development can be viewed from the following three aspects: 1) our service performance modeling methodology, 2) the developed ASBS, especially the QoS M/A modules, and 3) SOA-compliant simulations. For 1), it is difficult to perform large-scale experiments with many experimental conditions to cover many possible situations in actual systems due to resource constraints. Hence, we use SOA-compliant simulations to validate the ASBS design under conditions that cannot be examined in

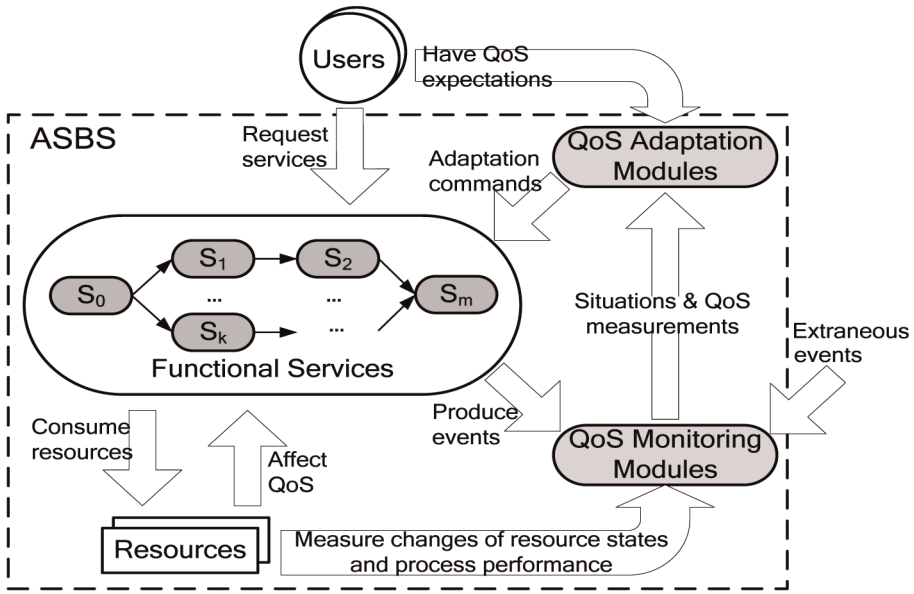


Fig. 1. A conceptual view of ASBS.

small-scale experiments with actual service implementations. In our SOA-compliant simulations, basic SOA elements, such as services, service registry and messages, as well as service QoS can be simulated to allow the construction of a simulated ASBS, which provides I/O and state data of simulated services to QoS monitoring modules [17]. Dynamic structures are supported in our SOA-compliant simulations, and enable runtime adaptation to system behavior by QoS adaptation modules. With these features, our SOA-compliant simulations support rapid construction and validation of simulated ASBS with simulated services and actual QoS M/A modules. For 2) and 3), we are currently evaluating the runtime performance with respect to the size of our QoS monitoring modules and the simulations developed using our new DEVS-Suite Simulator [18].

4 Activity-State-QoS Models for the Cause-Effect Dynamics in ASBS

The QoS M/A modules shown in Fig. 1 are the core of ASBS because they provide the capabilities for measuring the QoS aspects and system status, and making decisions for adapting service configurations in ASBS. However, these modules cannot be developed without knowing the QoS-related cause-effect dynamics in ASBS, especially the following:

1. *QoS composition*: How is the overall QoS of an ASBS determined by the QoS of individual services and the composition patterns used to compose the ASBS?
2. *QoS tradeoff*: How does one aspect of QoS of an ASBS affect other aspects of QoS?
3. *Environment and configuration impact*: How does the QoS of a service vary when the configuration of the service or the status of the execution environment changes?

Hence, it is necessary to have a systematic way to identify the underlying cause-effect dynamics driving the performance and tradeoffs among various QoS aspects. Fig. 2 depicts the cause-effect chain of user activities, system resource states and QoS performance, consisting of a service request from a user call for a system process that will utilize certain resources and change the states of the resources in the system environment. The changes of the resource state in turn affect the QoS of the process. The performance models revealing such cause-effect dynamics in SBS are called the *Activity-State-QoS (ASQ)* models. Constructing ASQ models is challenging because the differences in the types of services, the ways of service composition, and even service implementations, will affect different resource consumptions which in turn produce different characteristics of various aspects of QoS and the tradeoffs among the QoS aspects. In Sections 4.1-4.3, we will present the formulation of ASQ models, the design of experiments for collecting data related to user activities, system resource states, and service QoS, and our data analysis methodology for establishing ASQ models. We will also use an experiment with a voice

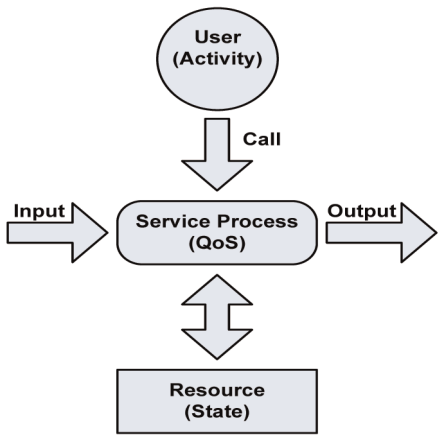


Fig. 2. The cause-effect chain of activity state QoS in SBS.

TABLE 1
Metrics for Accuracy, Timeliness, and Throughput

QoS Aspects	Metrics	Definition
Accuracy	Loss rate	The number of bits lost between two points in telecommunications after transmission.
	Error rate	The bit error rate is the frequency of erroneous bits between two points in telecommunication after transmission.
Timeliness	Service delay	The time elapsed between a user submitting a service request and receiving a service response.
	Network transmit delay	The time elapsed between the emission of the first bit of a data block by the transmitting end-system, and its reception by the receiving end-system.
Throughput	Data rate	The rate in which data are encoded.
	Bandwidth	The data transfer rate, measured in bits per second.

communication service as an example, and analytical results of the experimental data to illustrate these results.

4.1 ASQ Models

An ASQ model is a 7-tuple $<A, S_0, S, Q, R_{AS}, R_{AQ}, R_{SQ}>$, where A is a set of possible user activities that can be performed on a particular service, S_0 is a set of initial system resource states, S is the set of all possible system resource states, Q is the set of possible value for a particular aspect of QoS, R_{AS} is a relation defined on $A \times S_0 \rightarrow S$ representing how user activities and initial system resource states affect future system resource states, R_{AQ} is a relation defined on $A \times S_0 \rightarrow Q$ representing how user activities and initial system resource states affect Q, and R_{SQ} is a relation defined on $S \times S \rightarrow Q$ representing how changes of system resource states affect Q.

For each service in SBS, an ASQ model needs to be constructed for each QoS aspect of interest to the users and designers. All the ASQ models for various QoS aspects of all services in SBS collectively show the cause-effect dynamics in SBS. In SBS, user activities performed on a particular service are carried out through service invocations with various service parameters that will lead to different resource consumptions and thus different QoS performance. Since these parameters reflect various levels of user activities leading to different QoS performance, these parameters form the set of A variables in ASQ models.

A system resource state is represented by a set of state variables, where a state variable is a property of a certain system resource that can be directly acquired or derived from system monitoring, such as committed memory and CPU utilization.

To evaluate a QoS aspect, proper metrics for the QoS aspect need to be defined. There are well-defined metrics for timeliness, accuracy, and throughput [4] based on which analytical models can be developed to evaluate these three QoS aspects based on user activities and current system resource states of the system. Table 1 shows the metrics used for timeliness, accuracy, and throughput in ASQ models. These metrics are selected based on the given multimedia services and applications. For other services and applications, different metrics may be used. For instance, the accuracy of a web search service will be measured by the precision and recall of the search results, rather than the loss rate and error rate in Table 1, where recall is the percentage of web documents relevant to a query retrieved by the search service.

In Section 4.3, we will discuss how R_{AS}, R_{AQ}, and R_{SQ} are determined.

4.2 Our Experiments for Collecting Data to Build ASQ Models

To design an experiment for collecting data to build ASQ models, we need to construct an application scenario that uses one or more services, develop workload generation components to generate service requests and create different operational environments for the experiment, and then develop a data collection component to collect necessary data for constructing ASQ models. In the following, we will briefly describe our experiments, and use an experiment on the voice communication service as an example.

4.2.1 Selection of Services and Constructions of Application Scenarios for Experiments

To construct application scenarios for our experiments, we have selected a set of three multimedia services shown in Table 2 because these services have well-understood QoS requirements [4] and different resource consumption characteristics. Using these three services, the following four application scenarios are constructed for our experiments: online radio broadcasting, video sharing, motion analysis, and real-time surveillance camera. These four scenarios cover different service composition patters to construct ASQ models for the three selected services

TABLE 2
Services Selected for Our Experiments

Services	Functional Descriptions	Resource Consumption Characteristics
Voice communication	Provide voice-based communication over the network	Communication-intensive
Video streaming	Provide the live video feed from a web camera	Communication-intensive
Motion detection	Identify the differences in two images	Computation-intensive

TABLE 3
Service Parameter Settings of Our Experiments

Activity Variables	Level 1	Level 2	Level 3	Level 4	Level 5
Sampling rate (S)	44.1KHz	88.2KHz	132.3KHz	176.4KHz	220.5KHz
# of subscribers (C)	1	2	3	4	5
Buffer size (B)	16KBytes	24Kbytes	32KBytes	40KBytes	48Kbytes

individually as well as the compositions of these services. For example, the scenario online radio broadcasting only uses the voice communication service. The scenario real-time surveillance camera uses a sequential composition of the services video streaming and motion detection.

Besides different service composition patterns, we also considered various QoS requirements and tradeoffs covered by these four application scenarios. For example, according to [4], online radio broadcasting requires the delay (timeliness) to be less than 150 ms, error rate (accuracy) less than 0.1 percent, and data rates (throughput) larger than 56 Kbps. In the online radio broadcasting scenario, there may be many users with different minimum data quality requirements. Given limited system resources, it is necessary to adapt the configuration of online radio broadcasting to provide audio data with different quality to ensure that users' QoS requirements are satisfied while serving more users with improved audio quality.

4.2.2 Workload Generation and Identification of Experimental Control Variables

To construct an ASQ model $<A, S_0, S, Q, R_{AS}, R_{AQ}, R_{SQ}>$, the experiments under various conditions representing different user activities (A) and system resource states (S_0, S) need to be conducted to collect the necessary data for determining R_{AS}, R_{AQ}, and R_{SQ}. These experimental conditions are specified by a set of experimental control variables, which determine the system resource states, such as the current CPU workload and network bandwidth utilization, and certain characteristics of user activities, such as the number of concurrent users for a service and the quality of audio or video streams requested by users. These control variables are identified from the parameters of the selected services and the workload generation component to be used in the experiments. To generate workload in our experiments, we have developed an ExperimentLauncher to generate a specified number of service requests within a specified period of time to the selected services in our experiments. We have also developed a tool to generate controllable network traffic to simulate various network conditions.

4.2.3 Data Collection in Our Experiments

Data to be collected in our experiments reflect the system resource states and the three QoS aspects. In our experiments, we used IIS on Windows XP as our service platform, and developed a system monitoring tool using Windows Performance Objects to collect data of system status. We also added instrumentation code in the services used in our experiments to collect data for measuring the three QoS aspects of the services. Data were collected periodically and aggregated in two log files: one for the data collected from Windows Performance Objects, and the other for the data collected from the services.

4.2.4 Design of an Example Experiment on the Voice Communication Service

The example experiment is based on the online radio broadcasting scenario, in which multiple subscribers could simultaneously use the voice communication service to receive real-time voice streams of various qualities specified by the subscribers. We developed the voice communication service by converting an open-source Video Conferencing software package (http://69.10.233.10/KB/IP/Video_Voice_Conferencing.aspx) into Web Services using C# in .NET environment. The experimental data were collected from the service provider and subscriber(s). In this experiment, 232 counters from nine performance objects, including Process, Memory, Physical Disk, System, IP, UDP, TCP, Server, and Web Services, were collected.

Three service parameters were used in the experiment: the sampling rate for recording the voice stream, the number of subscribers, and the size of buffer for storing the voice stream before network transmission. Table 3 shows various settings for these service parameters. Under each experimental condition, 60 data observations were recorded during a period of one minute. In addition, the same data were also collected for the no-service condition, in which the system was monitored when the service was not running.

Some counters from Windows performance objects record accumulated values of system state or performance that increase over time and/or depend on the order of running experimental conditions. For example, Working Set Peak from the Process object records the peak value of the size of the working set of a process. The value of this counter both increases over time and varies based on the past experimental conditions. We need data screening to identify and remove such performance counters since they cannot accurately reflect the impact of different experimental conditions on the system resource state and service performance. To collect data for data screening, we conducted two small-scale runs, each with three experimental conditions and two no-service conditions, as follows:

- Run 1:
  1. no-service condition,
  2. [(S, C, B): (44.1 KHz, 1, 16 KBytes)],
  3. (132.3 KHz, 3, 32 KBytes),
  4. (220.5 KHz, 5, 48 KBytes), and
  5. no-service condition;
- Run 2 with the reversed order of that in Run 1.

Then, we conducted a complete run of the experiment under all 125 conditions to collect the data used for determining the effects of the conditions and constructing ASQ models.

4.3 Data Screening and Construction of ASQ Models

The following data analyses were carried out to screen the data and construct ASQ models:

D1. Data screening using Mann-Whitney test [19] on the data collected from the two small-scale runs to identify and remove the counters whose values depend on the time and order of running experimental conditions.

D2. Data analysis and modeling using data from the complete run of the experiment with the following statistical data analyses:

- *D2.1.* ASQ relationship discovery and categorization through analysis of variance (ANOVA) and the Tukey's Honest Significant Difference (HSD) test [20].
- *D2.2.* Generating the ASQ relationship map capturing A-S (service activity and system resource state) and S-Q (system resource state and QoS) relationships.
- *D2.3.* ASQ modeling using Multivariate Adaptive Regression Splines (MARS) technique [21].

The results from *D2.3* constitute the ASQ models, and are used in the development of QoS M/A modules in ASBS. In the following, we will discuss each step of the above construction process and use the voice communication experiment as an example to illustrate our methodology.

4.3.1 Data Screening

Mann-Whitney test [19] is used to identify the counters, whose values vary based on the time and/or the order of running experimental conditions. Mann-Whitney test is chosen because it uses nonparametric statistics based on ranks of the data, and thus depends little on the probability density distribution of data [20]. For each small-scale run and each counter from the Windows performance objects, we conducted a Mann-Whitney test to compare each of the three service conditions with each of the two no-service conditions. If a counter has inconsistent behavior under the same condition in the two runs, the counter will be removed because the effect of the same condition on the counter should be consistent if the counter is not affected by the time and/or order of service conditions.

In our voice communication experiment, 106 of 232 counters remained after data screening and are considered for further data analysis.

4.3.2 Data Analysis and Modeling

In ***D2***, we consider the activity parameters (number of subscribers C, sampling rate S, and buffer size B in our voice communication experiment) as activity variables (A variables), and the remaining performance counters after ***D1*** as state and QoS variables (S and Q variables).

In *D2.1*, ANOVA [20] is first used to analyze the effects of the activity variables in various experimental conditions on each remaining counter. If the effects are statistically significant based on the ANOVA results, the Tukey's HSD test is performed to determine how different levels of the activity variables affect the state and QoS variables. Based on the Tukey's HSD test results, the qualitative A-S (between an A variable and an S variable) and S-Q (between an S variable and a Q variable) relationships are revealed and further categorized into a certain type of ASQ relationships. In *D2.2*, an ASQ relationship map is constructed by representing each A, S, or Q variable as a node and an A-S or S-Q relationship as a link between the nodes. In *D2.3*, each qualitative A-S and S-Q relationship represented in the ASQ relationship map constructed in *D2.2* are refined into a quantitative prediction model of the corresponding S variables or Q variables. Since A-S and S-Q relationships are mostly nonlinear, the MARS technique is used to build a nonlinear regression model for each relationship.

In our voice communication experiment, ANOVA revealed 28 performance counters that at least one of the three activity variables has significant effects on. The Tukey's HSD test further revealed how these 28 counters were affected by the activity variables. Based on the Tukey's HSD test results, the 28 performance counters were grouped into five categories of A-SQ relationships as shown in Table 4. Among these five categories, the inconsistent change patterns of the seven variables in category 5 indicated that these variables were not only affected by the voice communication service, but also significantly affected by other system activities. Hence, only the 21 variables in categories 1-4 were further analyzed. Among the 21 variables in categories 1-4, some variables provide similar information. For example, the five variables in categories 1 and 4 reflect the memory consumption, only from different angles (consumed and remaining) and with different units (byte, KByte, and MByte). After removing such redundancy, we identified five state variables and one QoS variable from the 21 variables for further analysis. Fig. 3 shows the ASQ relationship map constructed using these six variables. For each state or QoS variable in the ASQ relationship map, we build a regression model to capture the quantitative A-S or S-Q relationship. As an example, we show the regression model for the QoS variable (*Fragments Created/sec_IP*) here. In this regression model, Q_{FC} denotes the value of *Fragments Created/sec_IP*. S_{IO}, S_{TC}, and S_{CB} denote the value of *IO Other Operations/sec_Process, Thread Count_Process*, and *Committed Bytes_Memory*.

$$
\begin{aligned}
(^{*})Q_{FC} = {} & 969.7692 - 0.1815 \,{}^{*}\max(0, S_{IO} - 1125.777) \\
& - 1.3904 * \max(0, 1125.777 - S_{IO}) - 87.4388 \\
& * \max(0, S_{TC} - 30) - 9.8643 \,{}^{*}\max(0, 30 - S_{TC}) \\
& + 93.5347 \,{}^{*}\max(0, S_{TC} - 37) + 1.5414e - 05 \\
& {}^{*}\max(0, S_{CB} - 490{,}037{,}200) + 3.7299e - 05 \\
& {}^{*}\max(0, S_{CB} - 518{,}844{,}400) - 7.0875e - 06 \\
& {}^{*}\max(0, 518{,}844{,}400 - S_{CB}).
\end{aligned}
$$

The R square value for this regression model is 0.9787, which shows that this regression model accurately captures the S-Q relationships observed in our experiment.

5 Development of QoS M/A Modules

In various application domains, QoS M/A in SBS may have various objectives, such as optimizing resource utilization,

TABLE 4
Five Categories of A-SQ Relationships

Category	State and QoS Variables (*counter name_object name)*	
1. Increase with S, C,B	% Committed Bytes In Use_Memory	Committed Bytes_Memory
2. Increase with S and C, decrease with B	% Privileged Time_Process	% Processor Time_Process
	% User Time_Process	Context Switches/sec_System
	Datagrams/sec_UDP	Datagrams/sec_IP
	Datagrams Sent/sec_UDP	Datagrams Sent/sec_IP
	File Control Operations/sec_System	File Control Bytes/sec_System
	Fragmented Datagrams/sec_IP	Fragments Created/sec_IP
	IO Other Operations/sec_Process	IO Other Bytes/sec_Process
3. Increase with C, first increase and then decrease with B	Thread Count_Process	Page Faults/sec_Memory
4. Decrease with S, C and B	Available Bytes_Memory	Available KBytes_Memory
	Available Mbytes_Memory	
5. Inconsistent change with S, C and B, and strong interaction of S, C and B	% Registry Quota In Use_System	Avg. Disk sec/Transfer_Physical Disk
	Datagrams Received Delivered/sec_IP	Processor Queue Length_System
	Datagrams Received/sec_IP	Datagrams Received/sec_UDP
	Page Faults/sec_Process	

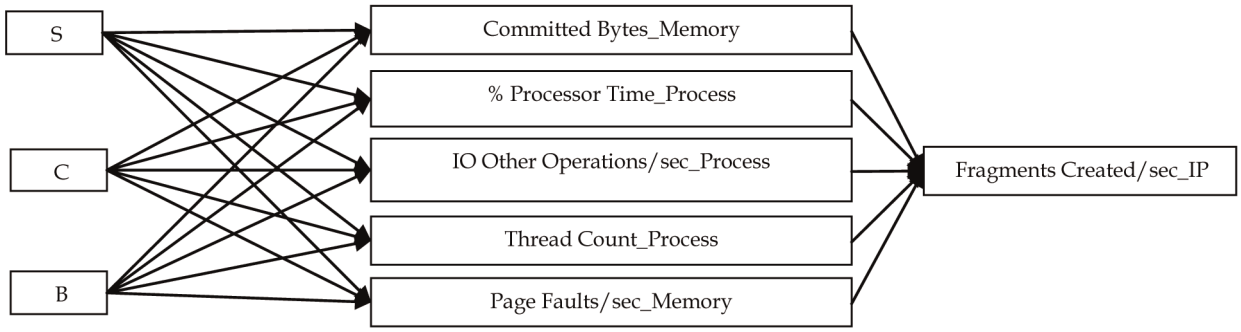


Fig. 3. The ASQ relationship map for our voice communication experiment.

maximizing service throughput, and improving service availability. Our research on the development of QoS M/A modules mainly focused on improving the overall QoS of applications involving long-running workflows (WFs) to satisfy users QoS requirements as much as possible. In this section, we will first present an overview of a model for distributed workflow execution, monitoring, and adaptation in SBS, and discuss how various software entities in this model interact among them to support QoS M/A in ASBS. Then, we will discuss how QoS monitoring modules in ASBS are developed based on the ASQ models generated in Section 4. Finally, we will also discuss some research ideas on the development of QoS adaptation modules in ASBS.

5.1 A Model for Distributed Workflow Execution, Monitoring, and Adaptation in SBS

Based on our work in [22], [23], we establish a model shown in Fig. 4 for distributed workflow execution, monitoring, and adaptation in SBS. In this model, functional services in SBS are provided by distributed hosts, which also provide various system services, including service discovery, performance monitoring, and resource management. Each host also provides a persistent data repository, in which service performance models (ASQ models) and data collected from system monitoring are stored. A workflow in SBS is coordinately executed by distributed WF agents, each of which is responsible for invoking one or more functional services. The status of each functional service used in the workflow is observed by a WF monitor, which checks whether the service QoS satisfies the user requirements. If the QoS of a functional service no longer satisfies the user requirements, the WF adaptors responsible for the functional service and the subsequent services to be invoked in the workflow adjust the configuration of these services or find alternative services to provide satisfactory QoS. The WF agents, monitors, and adaptors are deployed on distributed hosts, and communicate through named input/output channels supporting the publish/subscribe communication model.

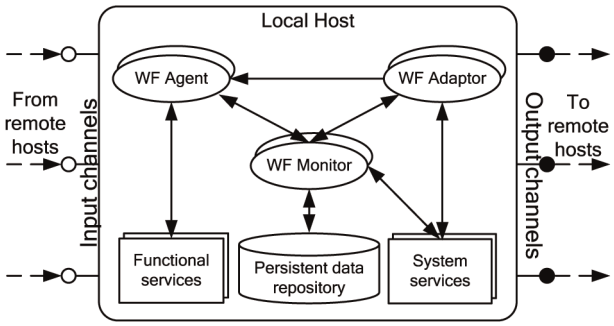


Fig. 4. A model for distributed workflow execution, monitoring, and adaptation in SBS.

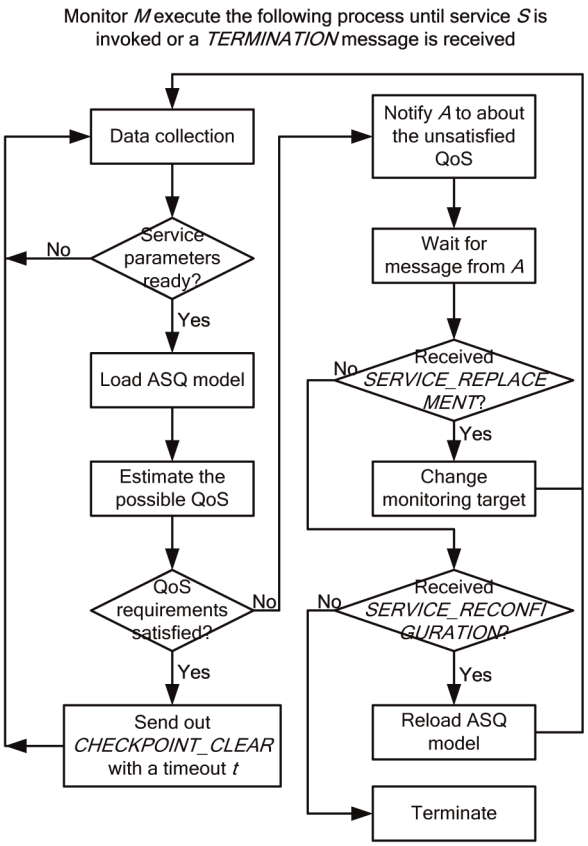


Fig. 5. Internal control flow of M during the preexecution phase.

5.2 Interactions Among WF Agents, Monitors, and Adaptors

Consider a functional service S deployed on a host H in a network N, and the WF agent W, monitor M, and adaptor A responsible for invoking, monitoring, and controlling S. To show how W, M, and A interact with each other to support QoS M/A, we will describe the internal control flows of W, M, and A during the preexecution phase (i.e., before S is invoked), and highlight the changes of the control flows during the execution phase.

Preexecution Phase. Before S is invoked, M, A, and W perform the following activities:

- As shown in Fig. 5, M first collects data for the status of S, H, and N. If the service parameters of S are available, M retrieves the appropriate service performance model for S, estimates the QoS of S based on the service parameters and the collected data, and checks whether the estimated QoS satisfies the QoS requirements for S. If not, M notifies A about the *SERVICE_REPLACEMENT* message from A, it changes the monitoring target to the new service replacing S. If M receives a *SERVICE_RECONFIGURATION* message from A, M retrieves the new service performance model from the persistent data repository of H and continues to monitor S. M repeats the above activities periodically until S is invoked or a *TERMINATION* message is received.
- As shown in Fig. 6, A waits for messages from M and other WF adaptors. If there are unsatisfied QoS requirements for S or unexpected delay in executing a service invoked before S in the workflow, A finds a new service configuration for S that can satisfy the QoS requirements or reduce the expected delay for executing S. If a new service configuration is found, A sends a *SERVICE_RECONFIGURATION* message to M and W. If not, A finds a replacement for S that can satisfy QR or reduce the expected execution delay. If such a replacement is found, A sends a *SERVICE_REPLACEMENT* message to M and W. Otherwise, A sends out a *TERMINATION* message and terminates. A repeats the above activities until S is invoked or A needs to terminate.
- As shown in Fig. 7, W waits for the service parameters of S, and messages from M and A. If the service parameters of S are ready, W sends M the service parameters. If a *CHECKPOINT_CLEAR* message is received from M, W checks whether all the services, which must finish before S starts. If yes, W

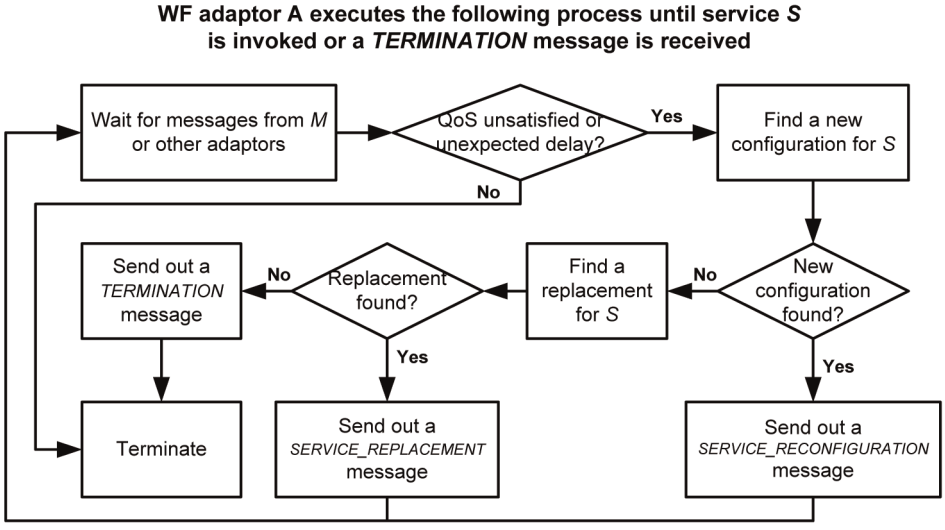


Fig. 6. Internal control flow of A during the preexecution phase.

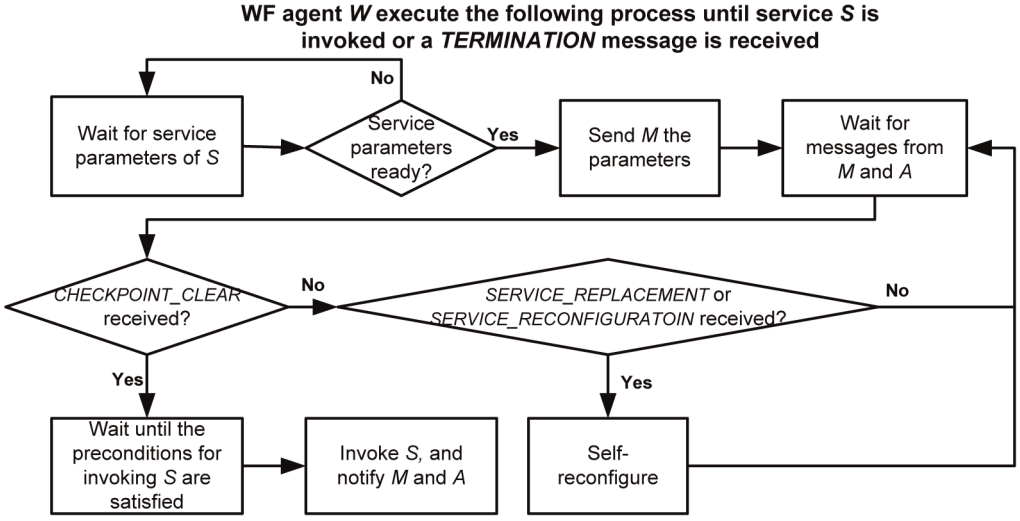


Fig. 7. Internal control flow of W during the preexecution phase.

invokes S immediately. After invoking S, W notifies M and A to enter the execution phase. If a *SERVICE_REPLACEMENT* or a *SERVICE_RECONFIGURATION* message is received from A, it is self-reconfigured to use the new service or the new service configuration.

Execution Phase. During the execution of S, M, A, and W perform similar activities as the preexecution phase, except the following:

- M measures the actual QoS of S instead of estimating the possible QoS. In addition, M checks whether the current service delay is greater than the maximum acceptable delay of S. If yes, M notifies the WF adaptors responsible for the services to be invoked after S in the workflow about the unexpected delay in the execution of S.
- A notifies W to suspend the execution of S before trying to find a new service configuration or a replacement for S.
- W suspends the execution of S if A notifies W to do so. The execution of S will restart or resume when W receives a *SERVICE_REPLACEMENT* or a *SERVICE_RECONFIGURATION* message.

5.3 Development of QoS Monitoring Modules

We have developed an approach to automatically generating QoS monitoring modules for a workflow based on the workflow specification, including the services used in the workflow, the control flow structure of the workflow, and the QoS requirements for the workflow, and the ASQ models of the services to be used in the workflow. We assume that the supported configurations for each service used in the workflow are also known a priori. Our approach consists of the following three steps: *S1)* Decomposing the QoS requirements for the workflow into requirements for individual services used in the workflow. *S2)* Synthesizing α-Calculus [22] descriptions of the QoS monitoring modules. *S3)* Compiling the synthesized α-Calculus descriptions into executable code.

In this paper, we will mainly discuss *S2*. For *S3*, we modified the compiler we developed in [22]. For *S1*, we represent the QoS requirements as a 6-tuple,

$$<max\_loss,\ max\_error,\ max\_service\_delay,\ max\_network\_delay,\ min\_data\_rate,\ min\_bandwidth>,$$

based on the six metrics shown in Table 1. Since the ASQ models generated in Section 4 are formulas similar to (*), the range of possible QoS of a service can be calculated based on the ASQ models of the service given the supported service configurations, which specify the range of possible values of activity variables used in the ASQ models. Then, the decomposition of QoS requirements can be easily done based on the control flow structure of the workflow and the range of possible QoS of the services used in the workflow.

After the decomposition of QoS requirements is done, the QoS monitoring modules for the workflow will be generated using the following two templates to support the interactions between monitoring modules and WF agents as well as QoS adaptation modules during the preexecution phase (Lines 1-34), and the execution phase (Lines 35-61):

Template 1:

```
1   fix $MONITOR$_PreExec(tuple s_info, int t,
      tuple counters, tuple qr)=
2       ((time t.let tuple s_data($S_COUNTER_1$, ..., $S_COUNTER_m$)=
3           SystemService:performanceMonitoring
            (tuple counters->s_counters, null,
            tuple s_info)instantiate zero)
4       +
5       (tuple $SERVICE$_param(tuple params).
6           let int handle=ASQ:loadModel(tuple params,
            tuple s_info)instantiate zero)
7       +
8       boolean $SERVICE$_invoked(boolean invoked)
9       +
10        tuple $SERVICE$_adapt(tuple adapt)
11        +
12        tuple $WF$_control(tuple control)).
```

```
13      if (invoked != null and invoked == true)then
14        $MONITOR$_Exec(s_info, t, params,
          counters, qr)
15      else if (adapt != null) then
16        if (adapt->cmd ==
          "SERVICE_REPLACEMENT")then
17          $MONITOR$_PreExec(adapt->s_info,
            t, adapt->counters, adapt->qr)
18        else if (adapt->cmd ==
          "SERVICE_RECONFIGURATION")then
19          let int handle=ASQ:loadModel(tuple
            params, tuple adapt->s_info)instantiate
20            $MONITOR$_PreExec(adapt->s_info,
              t, counters, adapt->qr)
21        else zero
22      else if(control != null and control->cmd ==
        "TERMINATION") then
23        zero
24      else if (params != null) then
25        let tuple eq(float s_delay, ..., float bw) =
          ASQ:qosEstimation(int handle, tuples_data)
          instantiate
26          let boolean b = ASQ:qosCompare(tuple
            eq, tuple qr)instantiate
27            if(b == true)then
28              tuple $SERVICE$_control<
                "CHECKPOINT_CLEAR", int t>.
29              $MONITOR$_PreExec(s_info,t,
                counters,qr)
30            else
31              tuple $SERVICE$_mon<"QoS NOT
                SATISFIED", s_info, params, qr>.
32              $MONITOR$_PreExec(s_info,t,
                counters,qr)
33        else
34          $MONITOR$_PreExec(s_info, t,
            counters, qr)
```

Template 2:

```
35  fix $MONITOR$_Exec(tuples_info, intt,
    tuple params, tuple counters, tuple qr) =
36      ((time t.let tuple data($COUNTER1$, ...,
        $COUNTERn$) =
37        SystemService:performanceMonitoring
          (tuple counters->s_counters, tuple counters->
          q_counters,
38          tuple s_info)instantiate zero)
39      +
40      tuple $SERVICE$_adapt(tuple adapt)
41      +
42      tuple $WF$_control(tuple control)).
43      if (adapt != null) then
...       //Similar to lines 16-23
51      else
52        let tuple cq(float s_delay, ..., float bw) =
          ASQ:currentQoS(tuple data) instantiate
53          let boolean b = ASQ : qosCompare(tuple cq,
            tuple qr) instantiate
54            if (b == true) then
55              $MONITOR$_Exec(s_info, t,
                params, counters, qr)
56            else
57              tuple $SERVICE$_mon<
                "QoS NOT SATISFIED", s_info,
                params, qr>.
58              if(qr->MAX_DELAY<cq->
                s_delay) then
59                tuple $WF$_control
                  <"UNEXPECTED_DELAY",
                  s_info, cq->s_delay, qr->
                  MAX_DELAY>.
60                $MONITOR$_Exec(s_info,t,
                  params,counters,qr)
61              else $MONITOR$_Exec(s_info, t,
                params, counters, qr)
```

The above templates are described in α-Calculus [22], which is a process calculus capable of describing service invocations, control flow structures of processes, and communications among processes. $MONITOR$, $S_COUNTER$, WF, and $SERVICE$ are placeholders in the templates. During the synthesis of monitoring modules, these placeholders will be replaced by the identifiers of monitoring modules, performance counters, workflows, and services.

The following five services are used in the above templates:

1. *SystemService:performanceMonitoring* takes the identifiers of state counters (e.g., *s_counters* in line 2) and QoS counters (e.g., *q_counters* in line 38) and the description of the service to be monitored as inputs (e.g., *s_info* in line 3), and returns a tuple containing the values of the state and QoS counters as the result. The description of the service to be monitored includes the identifiers of the service, the host providing the service, and the network in which the host resides, and the supported configurations of the service.
2. *ASQ:loadModel* takes the parameters (see *params* in line 6) and description of the service to be monitored as inputs, loads the corresponding ASQ models from the persistent data repository (see Fig. 4) to dynamically instantiate an ASQ model interpreter, which will be used to estimate the possible QoS of the service to be monitored, and returns the handle (see *handle* in line 6) to the ASQ model interpreter.
3. *ASQ:qosEstimation* takes the collected data of the state counters (see *s_data* in line 25) and the handle to the ASQ model interpreter for the service to be monitored as inputs, and calculates the estimated service QoS as the result (see *eq* in line 25).
4. *ASQ:qosCompare* takes the estimated service QoS and the QoS requirements (see *qr* in line 26) as inputs, checks whether the QoS requirements can be satisfied, and returns true or false.
5. *ASQ:currentQoS* takes the collected data of the state and QoS counters (see *data* in line 52) as the input, and calculates the current service QoS as the result based on the metrics in Table 1.

Notice that the above services are only interfaces for the actual service implementations because we want to keep the synthesized α-calculus descriptions platform-independent. The compiler we developed for the QoS monitoring modules can translate the invocations of the above services to the platform-dependent code. The actual service implementations have been done on Windows XP. In particular, *ASQ:loadModel*, *ASQ:qosEstimation*, *ASQ:qosCompare*, and *ASQ:currentQoS* are implemented according to our ASQ models and QoS metrics. *SystemService:performanceMonitoring* is implemented using Windows Performance Objects.

Line 1 declares a process named $\$MONITOR\$\_PreExec$, which takes four inputs: information of the service to be monitored, interval of data collection, identifiers of the counters to be monitored, and the QoS requirements of the service. The process $\$MONITOR\$\_PreExec$ is the sequential composition (denoted by ".") of two subprocesses, defined in lines 2-12 and lines 13-34, respectively.

The subprocess in Lines 2-12 is the composition of five subprocesses using nondeterministic choice (the "+" in lines 4, 7, 9, and 11), which means that at least one of the five subprocesses will be executed in runtime before executing the subprocess in lines 13-34. The five subprocesses are defined as follows:

- Lines 2-3. *time t* in line 2 is a time-out process, which puts the monitoring module into sleep for t seconds. The remaining part of lines 2-3 is an invocation of *SystemService:performanceMonitoring* and assigns the result to a tuple s_data. Tuple is a structured data type allowing multiple variables with various data types to be grouped together. Access to a variable in the tuple is represented by *TupleName->VariableName*, such as $counters{-}{>}s\_counters$ in line 3.
- Lines 5-6. $tuple\ \$SERVICE\$\_param(tupleparams)$ in line 5 defines an input action for accepting the parameters of $\$SERVICE\$$ through an input channel named $\$SERVICE\$\_param$. After receiving the service parameters, *ASQ:loadModel* will be invoked in line 6 to load the corresponding ASQ models for $\$SERVICE\$$.
- Line 8 defines an input action for receiving the notification of the invocation of $\$SERVICE\$$.
- Line 10 defines an input action for receiving adaptation commands.
- Line 12 defines an input action for receiving workflow control message.

The subprocess in lines 13-34 checks for various conditions following the process described previously (see Fig. 5) for the interaction between QoS monitoring modules and WF agents as well as QoS adaptation modules. Lines 35-61 define a process to be executed by a monitoring module during execution phase, which is quite similar to lines 1-34.

5.4 Development of QoS Adaptation Module

Our main idea for the development of QoS adaptation module is to first formulate QoS adaptation in ASBS as a multiobjective optimization problem [24]. We will define QoS expectation functions to quantify gains and losses for satisfying and violating users' QoS requirements, respectively. The ASQ models and QoS expectation functions will be used as the constraints and objective functions, respectively. To solve this optimization problem, we will develop optimization and control synthesis algorithms for generating appropriate adaptive control commands that adapt the system and service configurations to achieve the desired QoS tradeoffs. Then, the QoS adaptation modules in ASBS will be developed based on the optimization and control synthesis algorithms.

6 Validation of ASBS Design through SOA-Based Simulations

Simulation modeling offers important and unique capabilities for analysis and design of service-oriented computing systems that must satisfy multiple, competing QoS requirements. In order to aid design of SBS, it is important to employ a suitable modeling framework that can account for the SOA concepts. Based on the DEVS framework [25] and SOA concepts, a novel set of generic SOA-compliant simulation models are developed.

6.1 SOA-Compliant DEVS Modeling

The DEVS is a system-theoretic approach to specifying dynamical systems. It provides two canonical model types, called atomic and coupled models. An atomic model of a system can be defined in terms of input, output, state, and time sets with functions that determine next states and outputs, given current states and inputs at arbitrary time instances. Together, external, internal, confluent, output, and time advance functions define a component's behavior over time. A coupled model is defined in terms of atomic and/or coupled models. As in an atomic model, a coupled model contains a set of inputs, a set of outputs, a set of component names, a set of components, and a set of coupling relationships among the input and output ports of the composed model components. Atomic and coupled models interact with one another using messages (called entities) that are exchanged via input and output ports.

The SOA framework has a higher level of abstraction compared to the DEVS framework. The basic SOA elements include services, service registry, service discovery, and messages. We have defined a set of SOA-compliant DEVS (SOAD) elements [17] that correspond to the SOA (see Table 5). The SOAD models communicate with messages that represent service description, look up, and service messages. Runtime service registry and discovery are defined via an executive model. Given a set of services, it can handle other services to be dynamically added or conversely one or more existing services to be removed. The SOAD approach supports the composition of publisher, subscriber, and broker services to create flat and hierarchical compo site services. Furthermore, the SOAD modeling framework can be extended to support runtime composition of services using the DEVS dynamic structure modeling. Services communicate with one another via messages that contain service description or other content consistent with a chosen messaging framework. For example, a message from the broker to the subscriber is a service description which contains an abstract definition (an interface for the operation names and their input and output messages) and a concrete definition (consisting of the binding to physical transport protocol, address or endpoint, and service). Another message could be from a publisher to a subscriber. The implementation of these messages can be based on SOAP. To support the essential capability for simulating

TABLE 5
SOA Model Elements to the DEVS Model Elements Mapping

SOA Model Elements	SOAD Model Elements
services (publisher, subscriber, broker)	atomic models (publisher, subscriber, broker)
service description	entity (service-information)
messages	entity (service-lookup & service-message)
messaging framework	ports & couplings
service registry & discovery	executive model
service composition	coupled models (primitive and composite)

SBS, the SOAD modeling approach supports simulation of hierarchical service compositions. A hierarchical service composition has a publisher or subscriber service which itself is composed of other (flat or hierarchical) services. Since broker service is required for both primitive (flat) and composite (hierarchical) service compositions, either a single broker service or multiple broker services may be used. The DEVS model couplings support a single broker for primitive and composite service compositions. The common concept of DEVS and SOA modularity allows creating composite service composition without restrictions.

Generic SOAD simulation models are designed and introduced into the object-oriented DEVS-Suite simulator, a new extension of the object-oriented DEVSJAVA Tracking Environment [18]. The DEVS-Suite simulator supports parallel, efficient execution of the above SOAD models. The simulator affords animation of simulation execution and visualization of simulation results as time trajectories. With this simulator, common variations of SBS can be systematically modeled using SOA concepts. The simulator can address the difficulties associated with creating a real testbed and running extensive experiments (e.g., a system having many hundreds of services). The fundamental architecture and high-level design of software-based systems can be simulated and validated before low-level design, implementation, and testing.

6.2 Service-Based Software System Modeling and Simulation Results

To show the use of service-based software system modeling, a voice communication service described in Section 4.2 is considered. The voice communication application service, user services, and the broker service are modeled based on the actual services implemented in .Net. A model of a simple network with configurable delay and throughput is also developed. The network model is an abstraction of the communication links and routers/switches that allow the users to subscribe to the voice communication service via the broker. Each publisher, subscriber, broker, and network model has a corresponding transducer model. These transducer models are developed for systematic collection and analysis of simulation data (e.g., the amount of data that is transmitted among services as well as data transmission delays). These data sets are important for evaluating the throughput and timeliness of a system.

After constructing the simulated voice communication service, we conducted simulated experiments similar to the example experiment shown in Section 4. Fig. 8 shows that the instantaneous throughput of the simulated and the actual voice communication services where similar stochastic behavior is observed. Fig. 9 shows that the average throughput of the simulated service is similar to that of the actual service, when sampling rates are varied. The dynamics of the actual service in terms of its individual parts and their different compositions can be studied at a desirable level of detail using SOA-compliant simulated voice communication. Fig. 10 shows a simulation scenario, where two different sampling rates are requested by the five subscribers. The resultant dynamics show the impact of background traffic on system delay QoS for subscribers due to the decrease in network bandwidth with increasing background traffic. Therefore, the dynamics of ASBS designs

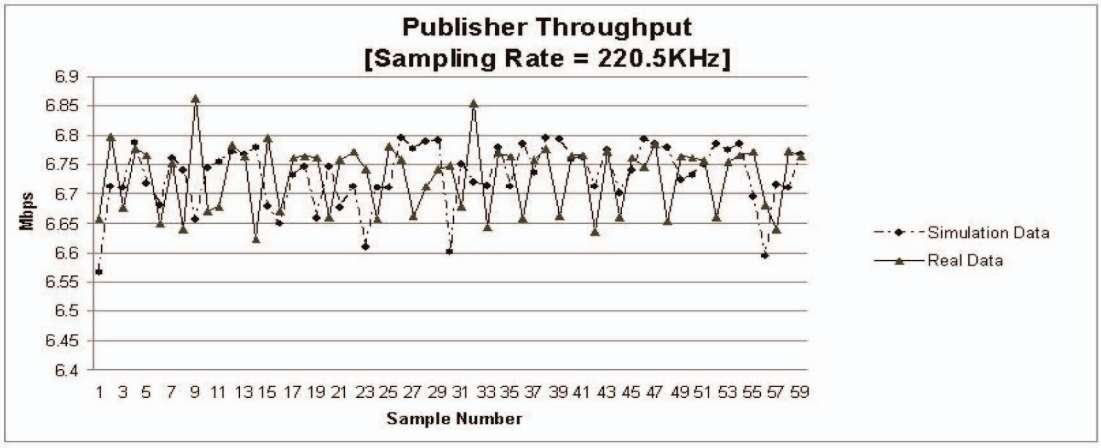


Fig. 8. Instantaneous throughputs of the simulated and real publisher.

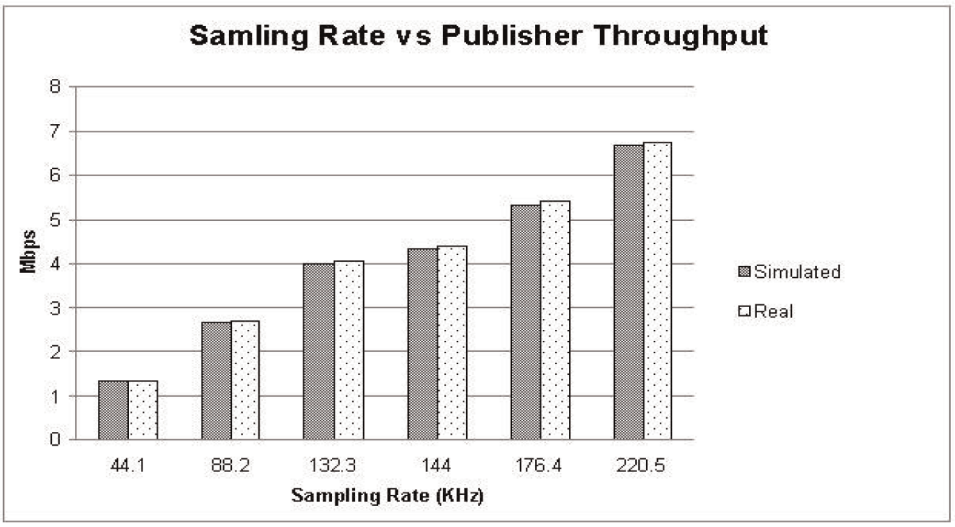


Fig. 9. Average throughputs of the simulated and real publisher for different sampling rates.

can be developed with direct support for measuring their QoS attributes like timeliness, throughput, and accuracy.

6.3 Validation Using Simulation

The SOAD modeling framework with the DEVS-Suite simulator offers a foundation for creating and simulating SBS. These simulation models help validating design of ASBS. With simulated services, we are able to 1) design and verify the structure and behavior of SOA-compliant service application models in terms of their time-based QoS attributes, 2) conduct simulation scenarios and validate design models against real experimental settings, and 3) use validated simulation models with real M/A modules to support rapid design and validation of ASBS. Simulation of the composite services is necessary for the design and development of the M/A modules.

7 Conclusions and Future Research

In this paper, we have presented our ideas toward the development of ASBS. We first presented our methodology for constructing ASQ models, and the summary of the results from our experiment based on a voice communication service to illustrate this methodology. We have also presented how QoS monitoring modules in ASBS can be developed based on the ASQ models, and the main idea on the development of QoS adaptation modules in ASBS. Finally, we have discussed our SOA-compliant DEVS modeling technique, which can support rapid design and validation of ASBS. Our approach will be extended to fully address various issues in developing ASBS with multiple QoS requirements, such as incorporating the impact of external events on various QoS aspects (especially security) in our ASQ models, and developing a performance index to provide a unified metric for multiple QoS performance of ASBS. In addition, we will improve our data analysis methodology for constructing ASQ models, develop algorithms for tradeoffs among multiple QoS aspects and methods for solving the optimization problem for QoS adaptation, and evaluate and improve the scalability of our approach.

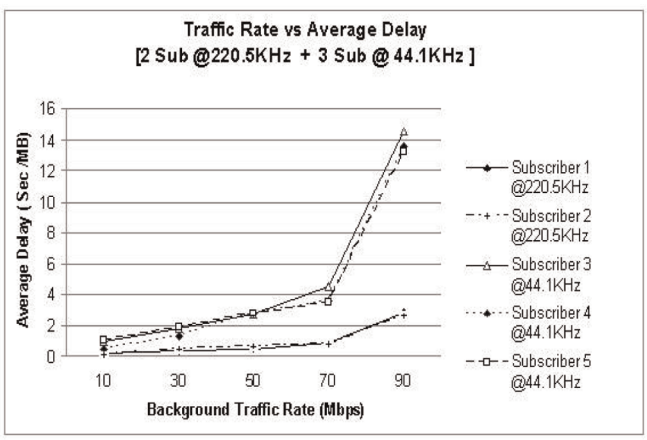


Fig. 10. Average system delay of the simulated and real subscriber.

Acknowledgments

This work is supported by the US National Science Foundation under grant number CCF-0725340.

References

[1] M. Reisslein, K.W. Ross, and S. Rajagopal, "A Framework for Guaranteeing Statistical QoS," *IEEE/ACM Trans. Networking*, vol. 10, no. 1, pp. 27-42, Feb. 2002.

[2] X. Xiao, T. Telkamp, V. Fineberg, C. Chen, and L.M. Ni, "A Practical Approach for Providing QoS in the Internet Backbone," *IEEE Comm.*, vol. 40, no. 12, pp. 56-62, Dec. 2002.

[3] Z. Yang, N. Ye, and Y.-C. Lai, "QoS Model of a Router with Feedback Control," *Quality and Reliability Eng. Int'l*, vol. 22, no. 4, pp. 429-444, June 2006.

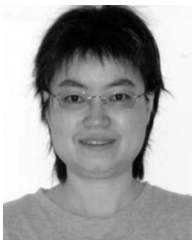
[4] Y. Chen, T. Farley, and N. Ye, "QoS Requirements of Network Applications on the Internet," *Information-Knowledge-Systems Management*, vol. 4, no. 1, pp. 55-76, 2004.

[5] J. Zhou, K. Cooper, I. Yen, and R. Paul, "Rule-Base Technique for Component Adaptation to Support QoS-Based Reconfiguration," *Proc. Ninth IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing*, pp. 426-433, May 2005.

[6] J. Jin and K. Nahrstedt, "On Exploring Performance Optimization in Web Service Composition," *Proc. ACM/IFIP/USENIX Int'l Middleware Conf.*, pp. 115-134, Oct. 2004.

[7] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villani, "An Approach for QoS-Aware Service Composition Based on Genetic Algorithms," *Proc. Conf. Genetic and Evolutionary Computation,* pp. 1069-1075, June 2005.
[8] C. Guo, M. Cai, and H. Chen, "QoS-Aware Service Composition Based on Tree-Coded Genetic Algorithm," *Proc. 31st Ann. Int'l Computer Software and Applications Conf. (COMPSAC '07),* pp. 361-367, July 2007.
[9] N. Kandasamy, S. Abdelwahed, and J.P. Hayes, "Self-Optimization in Computer Systems via On-Line Control: Application to Power Management," *Proc. First Int'l Conf. Autonomic Computing,* pp. 54-61, May 2004.
[10] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao, "AUTONOMIA: An Autonomic Computing Environment," *Proc. IEEE Int'l Conf. Performance, Computing, and Comm.,* pp. 61-68, Apr. 2003.
[11] D.R. Hild, H.S. Sarjoughian, and B.P. Zeigler, "DEVS-DOC: A Modeling and Simulation Environment Enabling Distributed Codesign," *IEEE Trans. Systems, Man, and Cybernetics, Part A,* vol. 32, no. 1, pp. 78-92, Jan. 2002.
[12] J.S. Dahmann, "High Level Architecture for Simulation," *Proc. First Int'l Workshop Distributed Interactive Simulation and Real-Time Applications (DIS-RT '97),* pp. 9-14, Jan. 1997.
[13] OPNET, "OPNET Modeler," http://opnet.com, Dec. 2005.
[14] D. Huang and H. Sarjoughian, "Software and Simulation Modeling for Real-Time Software-Intensive Systems," *Proc. Eighth IEEE Int'l Symp. Distributed Simulation and Real-Time Applications,* pp. 196-203, Oct. 2004.
[15] W.T. Tsai, Y. Chen, R. Paul, X. Zhou, and C. Fan, "Simulation Verification and Validation by Dynamic Policy Specification and Enforcement," *SIMULATION,* vol. 82, no. 5, pp. 295-310, May 2006.
[16] S.S. Yau, N. Ye, H. Sarjoughian, and D. Huang, "Developing Service-Based Software Systems with QoS Monitoring and Adaptation," *Proc. 12th IEEE Int'l Workshop Future Trends of Distributed Computing Systems (FTDCS '08),* pp. 74-80, Oct. 2008.
[17] H.S. Sarjoughian, S. Kim, M. Ramaswamy, and S.S. Yau, "A Simulation Framework for Service-Oriented Computing," *Proc. Winter Simulation Conf.,* pp. 845-853, Dec. 2008.
[18] DEVS-Suite Simulator, http://sourceforge.net/projects/devs-suitesim, May 2009.
[19] H.B. Mann and D.R. Whitney, "On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other," *Annals of Math. Statistics,* vol. 18, pp. 50-60, 1947.
[20] D.C. Montgomery, G.C. Runger, and N.F. Hubele, *Engineering Statistic,* fourth ed. John Wiley, 2006.
[21] J.H. Friedman, "Multivariate Adaptive Regression Splines (with Discussion)," *Annals of Statistics,* vol. 19, pp. 1-141, Mar. 1991.
[22] S.S. Yau, H. Davulcu, S. Mukhopadhyay, H. Gong, D. Huang, P. Singh, and F. Gelgi, "Automated Situation-Aware Service Composition in Service-Oriented Computing," *Int'l J. Web Services Research (IJWSR '07),* vol. 4, no. 4, pp. 59-82, Oct.-Dec. 2007.
[23] S.S. Yau, D. Huang, and L. Zhu, "An Approach to Adaptive Distributed Execution Monitoring for Workflows in Service-Based Systems," *Proc. 31st Ann. Int'l Computer Software and Application Conf.,* vol. 2, pp. 211-216, July 2007.
[24] R.L. Rardin, *Optimization in Operations Research.* Prentice Hall, 1998.
[25] B. Zeigler, T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation,* second ed. Academic Press, 2000.

Stephen S. Yau received the PhD degree in electrical engineering from the University of Illinois, Urbana. He is the director of the Information Assurance Center and a professor of computer science and engineering at Arizona State University, Tempe. He was previously with the University of Florida, Gainesville, and Northwestern University, Evanston, Illinois. He served as the president of the IEEE Computer Society and the editor-in-chief of *Computer*. His current research is in distributed and service-oriented computing, adaptive middleware, software engineering, trustworthy computing, and data privacy. He is a fellow of the IEEE and a fellow of the American Association for the Advancement of Science.



Nong Ye received the PhD degree in industrial engineering from Purdue University. She also received the MS degree from the Chinese Academy of Sciences and the BS degree from Peking University, in computer science. She is a senior member of the IEEE and a professor of industrial engineering at Arizona State University (ASU), Tempe. Before joining ASU, she was an assistant professor of industrial engineering at the University of Illinois, Chicago. Her research interests are in information assurance, quality of service, and data mining.

Hessam S. Sarjoughian received the PhD degree in electrical and computer engineering from the University of Arizona, Tucson. He is an assistant professor of computer science and engineering at Arizona State University, Tempe. He is a codirector of the Arizona Center for Integrative Modeling & Simulation (ACIMS). His research focuses on modeling and simulation methodologies, agent-based simulation, visual simulation modeling, model composability, and codesign modeling. He is a member of the IEEE.

Dazhi Huang received the BS degree in computer science from Tsinghua University, China. He is a PhD student in computer science at Arizona State University, Tempe. His research interests include middleware, mobile and ubiquitous computing, and workflow systems in service-oriented computing environments.

Auttawut Roontiva received the BS degree from Chulalongkorn University in 2000 and the MS degree in mechanical engineering from Arizona State University (ASU), Tempe, in 2004. He is presently a PhD student in industrial engineering at ASU. His research interests include information system security and assurance, adaptive service-based software systems, and data mining.

Mustafa Gökçe Baydogan received the MS degree in industrial engineering from Middle East Technical University, Turkey, in 2008. He is a PhD student in industrial engineering at Arizona State University, Tempe. His research interests are data mining, heuristic search, and simulation.

Mohammed A. Muqsith is a PhD student in computer science at Arizona State University (ASU). He is a member of the Arizona Center for Integrative Modeling & Simulation at ASU. His research interests include simulation-based design, service-oriented software/hardware codesign, system performance modeling, software engineering, and networked embedded systems.