International Journal of Software Engineering
and Knowledge Engineering
Vol. 19, No. 5 (2009) 653–678

World Scientific
www.worldscientific.com

AN INTELLIGENT CONTROL ARCHITECTURE FOR ADAPTIVE SERVICE-BASED SOFTWARE SYSTEMS*

CHANG-HAI JIANG†, HAI HU‡ and KAI-YUAN CAI§,**

Department of Automatic Control,
Beijing University of Aeronautics and Astronautics,
Beijing 100191, China
†*jchbh@asee.buaa.edu.cn*
‡*huhai.orion@gmail.com*
§*kycai@buaa.edu.cn*

DAZHI HUANG¶ and STEPHEN S. YAU‖,††

Arizona State University,
Tempe, AZ 85287-8809, USA
¶*dazhi.huang@asu.edu*
‖*yau@asu.edu*

Service-oriented architecture (SOA) for distributed computing has become increasingly popular due to the big advantage that distributed applications can be rapidly synthesized with the needed services provided by various service providers through heterogeneous networks. Systems based on SOA are called Service-based Systems (SBS), and a special variety of SBS, namely the Adaptive Service-Based Systems (ASBS), is aimed to be adaptable to constantly changing user requirements, environments and resource constraints. An important and difficult issue is how to design and develop ASBS to satisfy multiple QoS requirements in an open dynamic environment. In this paper, inspired by the underlying principle of hierarchical intelligent control, a three-layer architecture for developing and deploying ASBS is presented to address this issue. Compared with existing architectures for SBS, the advantage of using our architecture is that it provides the flexibility for system designers to adopt different control based approaches to guarantee user requirements and satisfy resource constraints at different levels of the system. Moreover, our architecture enables the system to take hierarchical adaptation actions at runtime to avoid possible violation of user requirement or resource constraint. An example is given to illustrate how to adopt our architecture to guide the design of a simple

*This paper is a revised and extended version of the following paper: C. H. Jiang, H. Hu, K. Y. Cai, D. Huang and S. S. Yau, An intelligent control architecture for adaptive service-based software systems with workflow patterns, in *Proc. 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC'08)*, July 2008, pp. 824–829.

**Cai was supported by the National Science Foundation of China and Microsoft Research Asia (Grant No. 60633010).

††Yau's research was supported by the National Science Foundation under Grant No. CCF-0725340.

ASBS, and preliminary experimental data are presented to demonstrate the feasibility of developing ASBS based on our architecture.

Keywords: Service-oriented architecture; service-based systems; software cybernetics; intelligent control.

1. Introduction

The emergence of service-oriented architecture (SOA) has enabled great advances in distributed applications, such as web-service based applications [46], enterprise computing infrastructures, and Global Information Grid (GIG) [51]. Systems based on SOA, called service-based systems (SBS), can be rapidly composed with services provided by various service providers through heterogeneous networks, following certain workflows, which define how the execution of various services are coordinated to complete specific user tasks that cannot be done by individual services. A special variety of SBS, namely the Adaptive Service-Based Systems (ASBS), is aimed to be adaptable to constantly changing user requirements and situations reflecting the status of the environment, system resources, services, and user tasks [17, 52–54]. Figure 1 gives a conceptual view of the entities and interactions involved in ASBS. Note that the QoS Monitoring and Adaptation modules and the underlying functional services together form a closed loop control system [7]. For software engineers, the following two important questions in the design and development of ASBS need to be considered:

Q1: *How to design a SBS adaptable to changing user requirements, performance specifications and resource constraints?*

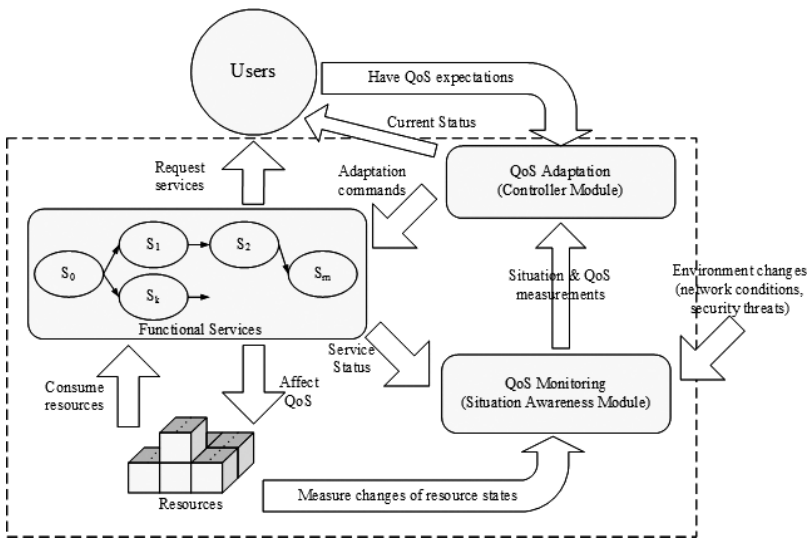

Fig. 1. The conceptual view of ASBS.

Q2: *How to detect or predict violations of timing and resource constraints for certain workflows in runtime, and adapt the system to overcome such violations?*

Following the principle of software cybernetics, which explores the interplay between software and control [6], in this paper we present a three-layer intelligent control architecture for developing and deploying ASBS to address the above questions. Furthermore, we adopt the principles of feedback control theory to develop controllers that provide runtime resource monitoring and workflow adaptation with multiple QoS requirements for each architectural layer. The structure of our architecture is inspired by the underlying principles of the hierarchical intelligent control theory [38]. Compared to existing architectures for SBS, the advantage of using our architecture is that it provides the flexibility for system designers to adopt different control theoretic based approaches to guarantee user requirements and satisfy resource constraints at different levels of the system. Moreover, our architecture enables the system to take hierarchical adaptation actions in runtime to avoid possible violation of user requirements or resource constraints. Such adaptations can appear at three different levels in ASBS:

- User level, where decisions on which user requests for services should be accepted are made dynamically based on situation changes;
- Workflow level, where different workflows may be selected to achieve the user goal according to changes in user requirements and situations;
- Service level, where service configurations may be changed to provide better QoS outputs.

The first benefit of using our architecture is that it enables us to address $Q1$ with the help of various state-of-the-art techniques based on feedback control theory which has been widely applied to the design of adaptive systems [1, 2, 50]. However, due to the scale and the unique characteristics of ASBS, such as loosely-coupled, late-binding, service/resource sharing, highly dynamic environments, etc., it is difficult to develop a centralized control system for runtime monitoring and services adaptation. Our architecture overcomes this difficulty by dividing the system into three hierarchical layers and adopting individual control schemes to monitor and control the QoS performance and resource constraints, considering the unique features and performance requirements for each layer respectively. For example, knowledge-based control approaches can provide flexible and adaptable management of the user request, whereas it might not be suitable for accurate control of the atomic services due to its complexity and cost. Therefore our architecture allows the engineers to adopt knowledge-based control approaches for the user management layer and used a simpler but accurate alternative, such as the adaptive feedback control scheme, to monitor and control the underlying services layer.

The second benefit of our architecture is its simplicity compared to other architectures developed by industrial practitioners and academic researchers [19, 44, 47]. By adopting the multi-layer structure, the complex interactions and activities

among various entities in SBS are organized hierarchically providing a clearer view of the whole system as well as the underlying details. Moreover, the resource constraints and environment variables can be monitored independently with the help of situation aware technologies. The context information on potential resource constraint violation and environmental changes, such as memory depletion and emerging security threat, are synthesized and forwarded to the controller of each architectural layer. Hierarchical adaptation actions are then conducted in a collaborative manner among the layers to avoid resource or security violations. This provides us with an effective approach to address $Q2$ in an open dynamic environment.

The rest of this paper is organized as follows: Sec. 2 presents a summary of the state-of-the-art in the related areas; Sec. 3 gives the detailed presentation of our three-layer intelligent control architecture for ASBS; more details on the underlying infrastructure of the components are illustrated through an example in Sec. 4; experiments are reported in Sec. 5 and conclusions are presented in Sec. 6.

2. Related Studies

In recent years, more and more efforts have been attracted to the research on how to design and adapt systems to satisfy various QoS requirements. The related research works mainly focused on the following five areas: QoS-aware service composition, integrated QoS control/management, self-tuning techniques, autonomic middleware, and software architecture. The architecture presented in this paper is closely related to Saridis's works on hierarchical intelligent control and intelligent machines [38]. This paper is also related to previous works in software cybernetics and feedback control in general.

QoS-aware service composition [8, 12, 18] aims to find optimal or sub-optimal service composition satisfying various QoS constraints, such as cost and deadline, within a reasonable amount of time. Various techniques were presented for QoS-aware service composition, such as service routing [18] and genetic algorithms [8, 12]. However, runtime adaptation of service composition cannot be efficiently handled by most QoS-aware service composition methods.

Integrated QoS control/management is to provide QoS support in a consistent and coordinated fashion across all layers of enterprise systems. In [11], the main requirements of the QoS control mechanisms are analyzed and a corresponding control mechanism to fulfill these requirements was presented to provide differentiated services as well as protection against server overloads. A QoS mapping framework for differentiated services was developed by Hwang *et al.* [15] to provide reliable and consistent end-to-end service guarantee for video streaming services. The control techniques adopted in these studies are good enough for optimizing a specific QoS requirement but insufficient for the overall design of an ASBS.

Self-tuning techniques and autonomic middleware providing support for service adaptation and configuration management was developed [10, 16, 20, 23]. In [20], an online control approach was presented to automatically minimize power

consumption of CPU while providing satisfactory response time. A self-healing middleware framework was presented in [13] to provide the self-healing properties for QoS management in web service-based distributed interactive applications. However, these techniques and middleware are not designed for SBS, which is more loosely coupled and more difficult to control since SBS is often composed by services provided by multiple organizations.

Another category of related studies tries to address the problem in the context of software architecture. Software architecture provides a global perspective on the software system rather than the implementation details and provides major benefits in designing large complex systems such as service-based systems [21]. Several efforts developed general architectures to guarantee QoS for service-oriented computing in the dynamic environment. The OASIS SOA Reference Model is an abstract framework which defines the essence of SOA. However the model is conceptual and does not cover the complicated SOA architecture style in applications. On the other hand, many SOA vendors such as IBM [3], Microsoft [28], BEA [5], and Oracle [30] propose their specific SOA model, but they are more development oriented rather than QoS oriented. Patel *et al.* [31] proposed a QoS oriented framework, namely WebQ, for web services selection, binding and execution. The architecture is comprised of several collaborative components that allow for QoS oriented adaptive management of web service based workflows, namely the *Web Service Mediator*, the *Task Execution Engine* and the *Expert System* (together with *Rule Repository* and *Knowledge Base*). However their study mainly focuses on the service selection and execution perspective while our architecture also considers user management and workflow management. Architectural patterns and blueprints for SOA have been used to develop better SBS for enterprise usage [43]. QoS management architectures [29, 41] were presented to achieve a trade-off between resource and QoS performance. An architecture for an autonomic computing environment (AUTONOMIA) was also presented in [10]. In this architecture, various middleware services for fault detection, fault handling, and performance optimization are included. Tang *et al.* [44] built a generic and abstract model of enterprise service-oriented architecture consisting of seven sets of components, including services, consumers, and service manager. Cheng *et al.* [9] presented a framework using software architecture models to analyze whether an application requires adaptation and how the adaptation should be conducted. A three layer architecture composed of the *Runtime Layer*, the *Model Layer* and the *Task Layer* was developed as the basis of system reconfiguration and adaptation at runtime [17]. The structure of the three layer architecture may look similar to the one we present in this paper, but the context of the approach is more general and do not involve the characteristics of SBS. Moreover, no feedback control mechanism is adopted while our architecture clearly defines the goal of control and the underlying feedback mechanism at each different layer in this three-layer architecture. Wang *et al.* [47] presented an integrated QoS management architecture in a Publish/Subscribe style for enterprise SBS, including an XML-based language to express QoS requirements as QoS messages, a set of the

supporting services for their architecture and developed and experiments to validate the feasibility. However, their architecture adopts a flat layout where components and external supporting services are loosely coupled, and may lead to difficulties in monitoring and controlling of the system performance. Raibulet *et al.* [33] presented the important components reflective layers of an adaptive and QoS based architecture for Adaptive Resource Management (ARM) to represent and manage QoS of the system. However, their work focused on describing the components rather than how the components interact and adapt to guarantee overall QoS. Jin *et al.* [19] presented a framework for agent-based service-oriented modeling of the structure, collaboration and behavior of the software system rather than designing QoS adaptive SBS.

Although several QoS adaptive architectures have been presented, they focus on specific aspects such as session control [55] or context information [41], and no design scheme from service composition to runtime adaptation. Moreover, a majority of current architectures do not incorporate feedback control mechanism into the design and runtime adaptation of the system, or only incorporate single layer architectures, which are not sufficient to address most of the control problems in SBS.

Feedback control theory was originally developed for physical process control. Its use in the context of software systems is novel. Software performance control presents a myriad of interesting challenges such as modeling software process as control systems, designing the proper sensors, controllers and actuators for the software system, and mapping computing problems into the feedback control domain. The main idea of this paper is inspired by notion of *software cybernetics*, which explores the interplay between control theory and software engineering [6]. Recently, there are many successful applications of this idea in the area of distributed computing [26]. For example, in [27] elements of control theory are applied in a Web caching context to guarantee a desired difference between the hit ration on different content classes. Feedback control theory is applied to achieve overload protection, performance guarantee and service differentiation in the presence of load unpredictability [1]. Feedback control theory has also been widely adopted to guarantee the Quality of Service (QoS) in web servers [1] and object middleware [2] in dynamic environments to provide differentiated services for users. An adaptive dual control framework that optimizes the tradeoff between the control goal and system identification for QoS design was presented in [25, 50].

In this paper we use feedback control approaches as basic building blocks of each architectural layer of the ASBS to achieve hierarchical control objectives and satisfy a wide range of performance requirements. Considering the unique characteristics of Service-based Systems, such as loosely-coupled, late-binding, service/resource sharing, highly dynamic environments, etc., it is difficult to develop a centralized control scheme for all aspects involved from service composition to runtime adaptation. Inspired by the idea of hierarchical intelligent control first presented by Saridis in 1977 [38], we will use a three-layer intelligent control architecture for adaptive service based systems. The original hierarchical intelligent control approach was

designed to organize, coordinate, and execute anthropomorphic tasks by a machine with minimum interaction with a human operator [36, 48]. A three-tier structure consisting of organization, coordination, and execution levels forms the architecture of an intelligent machine is proposed by Saridis and Valavanis [39]. Later, a unified model was presented by Varvatsoulakis *et al.* [45] for a class of intelligent machines, suitable for flexible manufacturing systems based on the theory of hierarchically intelligent control systems. Despite the difference in the application background and the mathematical techniques involved, the basic structure of three-level hierarchical system, including the organization, coordination and execution levels remains unchanged. In this paper, we will adopt the basic principles and structure of hierarchical intelligent control to address the control problems encountered in the design of ASBS.

Substantial research has been done on workflow planning and web service composition to construct and execute a workflow to satisfy users' functional requirements [4, 32, 34, 42, 49]. In this paper, we will focus only on the QoS requirements of ASBS. For a specific user task, there are usually various workflows with different workflow patterns that satisfy the goal. Workflow patterns address business requirements in an imperative workflow style expression, but do not depend on specific workflow languages [35].

3. Our Architecture

In this section, we will present the overall approach of the three-layer intelligent control architecture for ASBS. First we will explain the hierarchical intelligent control which will be used in our three-layer intelligent control architecture. In the subsequent subsections, we will present the architectural components and discuss their interactions among the architectural layers.

Our architecture will provide a global perspective on the software system rather than the implementation details. We will also provide an example in Sec. 5 to walk through the detailed designing process and explicitly explain how the controllers are synthesized.

3.1. *Three-layer intelligent control architecture for ASBS*

Figure 2 depicts the complete layout of the three-layer intelligent control architecture for adaptive SBS. The major components in our architecture are the hierarchical layers: the User Management layer, the Workflow Management layer, and the Service Management layer, respectively. Other entities in the system, including the environment and resource monitors as well as the underlying services, are also modeled. Then the input/output features of each layer are depicted by the connecting arrows. For example, the User Management layer receives user requests and QoS specification from the user and report service outputs and QoS status back to user. On the other hand the User Management layer sends out task requests and PI specification to the Workflow Management layer and retrieves task and resource

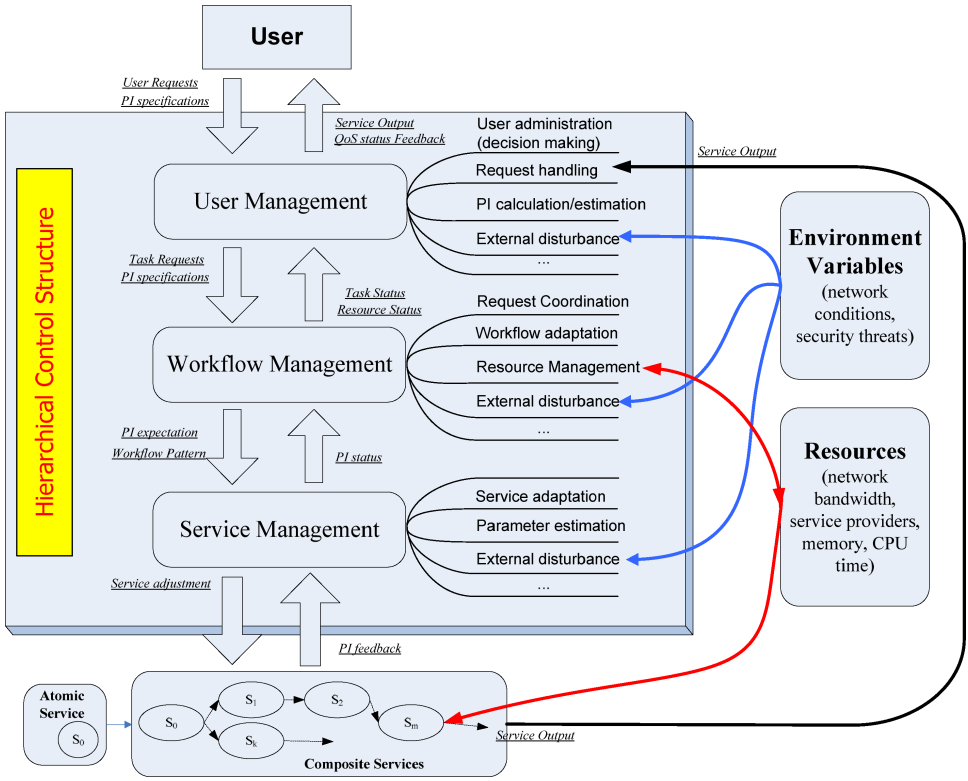

Fig. 2. Our three-layer intelligent control architecture for ASBS.

status from it. Similar observations can be made for other components in the architecture.

In order to synthesize controller modules in an adaptive SBS, a Performance Index (PI) needs to be defined for assessing the system performance. Note that the adaptive SBS needs to consider not only the QoS requirements but also many other factors such as resource constraints and security issues. The goal of the controller modules for an adaptive SBS is to optimize the overall PI rather than just QoS performance. In this study, PI is defined as a function of multiple QoS requirements, which is a special case. More general descriptions of PI can be found in [53]. In the following subsections, the tasks to be handled and issues to be considered when designing the controllers for each layer are investigated respectively.

3.2. *Components in our architecture*

3.2.1. *User management*

The major tasks of the User Management layer include request handling and user administration. More specifically, when a user request arrives, the User Management layer first needs to decide whether the request should be accepted or denied. Then

depending on the context of the user request, such as if the user is a premium subscriber or a normal subscriber, choose an appropriate PI specification for the task to handle the request. It also needs to consider the resource status and environment status when making the above decisions.

In this layer, the controller determines whether a user request should or should not be accepted based on the feedbacks such as service and resource status. When a user request is accepted, it creates a task together with a PI specification for the workflow management layer to handle the request. The PI specification is assigned according to the user context and the resource situation. External disturbance such as potential security threat will also be considered when assigning the PI specifications. Some requests might be shredded to ensure the QoS performance of the requests that are currently being handled. It calculates the PI of all concurrent user requests, and generates "PI not satisfied" events when the PI of certain user requests cannot be satisfied.

The User Management controller is the top level controller in the architecture and corresponds to the organization level in the hierarchical intelligent control theory. The organization level is intended to perform such operations as planning and high-level decision making from long-term memories. It may require high-level information processing such as the knowledge-based systems encountered in artificial intelligence [37].

3.2.2. *Workflow management*

The Workflow Management layer decides the workflow pattern to be used for each user request handling task, and coordinates the tasks to guarantee the PI specifications. Coordination actions include adjusting the priority of the tasks depending on their QoS and resource status. Moreover, the PI specification for each task is decomposed to specific PI expectations for each individual component in the workflow.

The controller for Workflow Management layer needs to select the optimal workflow pattern from a pool of candidate workflow patterns that complete the same user task. A prioritization algorithm is needed to prioritize the concurrent tasks at runtime to optimize overall PI. Moreover, the controller estimates environment variables such as network condition and update its workflow selection policy and prioritization algorithm online. This is critical for the ASBS to adapt itself to the dynamic environment.

The Workflow Management layer corresponds to the coordination level in the hierarchical intelligent control theory which requires less intelligence but rely more on the accurate estimation of system parameters and runtime update of its control policy. It plays a central role in the resource management of the ASBS since most of the workflow adaptation actions depend on the amount of available resource and the workflow pattern. For example, a parallel workflow pattern consumes multiple times of resource compared to a sequential pattern while it provides better QoS performance, the controller needs to balance the resource consumption and QoS performance when deciding which workflow pattern should be adopted.

3.2.3. *Service management*

The major task for the Service Management layer is to selects the most suitable services based on the chosen workflow pattern. Feedback information on service status and environment variables such as service provider status, network condition, and security level needs to be taken into account for decision making. For example, in an undesirable network condition where only very limited bandwidth is available, it is preferable to use an encoding service which provides higher compression ratio to reduce the transmission time for the encrypted content; if a service provider disconnects frequently, it is preferable to use another one with more stable connection.

In general, the controller in this layer finds and synthesizes the optimal services under the current situation based on an object function which intends to optimize the PI of a specific workflow. The parameters involved in the object function needs to be monitored or estimated online. Adaptation actions include service substitution and service parameter adjustments.

The Service Management layer corresponds to the execution level in the hierarchical intelligent control theory which incorporates lowest intelligence level and highest precision level. Similarly, the controller in the Service Management layer depends on the precise estimation and calculation rather than intelligent decision making techniques. Considering the number of services and users in real life SBS, the controller needs to be simple, accurate and reliable.

3.3. *Component interactions*

Despite the above mentioned ordinary I/O interactions between adjacent layers, the majority of the component interactions happen when the runtime environment is changing and the current system setting can no longer satisfy user requirements. Take security issues for example, when the environment monitor detects a potential security threat, such as eavesdropping, it raises a warning signal to the system and demand for an adaptation action to increase the security level. This is expressed as an update on the PI specifications where the weight of security is adjusted or a security constraint is added to the function. However adaptation actions, such as rekeying the data transactions, often consume system resource and may lead to deteriorated QoS. Then the controller on each layer needs to interact with each other to provide a solution that minimizes the negative impacts of adaptation.

First, the Service Management layer attempts to adjust its service parameters by selecting a more secure algorithm for data encryption and rekey all on-going transactions. If the estimated cost of rekeying will lead to unsatisfied PI, it will immediately inform the Workflow Management layer that the adaptation on the Service Management level is infeasible and ask for workflow adjustments. The Workflow Management layer receives the request and evaluates two possible solutions: change the prioritization of tasks and select an alternative workflow pattern for the tasks in the queue. If one of the solutions is expected to satisfy the PI, the corresponding

adaptation directives are sent to the Service Management layer for execution. On the other hand, if neither solutions works, the Workflow Management layer sends a signal to the User Management layer to notify the situation and ask for high-level intervention such as temporarily shutting down some of the services or enforce more strict load shredding for incoming user requests.

The above example gives a picture on the interactions among layers when there is an external disturbance in the environment. More scenarios can be investigated and the effective administration of the interactions can be an interesting and important future topic. Due to the space limit, we will not investigate the issue further in this paper.

3.4. *Remarks*

As mentioned before, in the hierarchical intelligent control theory, [38] a complex system is designed with three hierarchical levels of control: the organization, coordination, and execution levels [22, 40]. The control problem at each individual level is then handled with the appropriate control techniques ranging from artificial intelligence to tradition PID control schemes depending on the level of intelligence required for the specific task. The hierarchical intelligent control theory is based on the need for the development and utilization of machines that contain enough intelligence to perform autonomous tasks in uncertain environments [17] which resembles the problems we discuss in this paper.

For complex systems like ASBS, hierarchical intelligent control enables us to divide the system into hierarchical layers which can be designed individually. We first redefine the organization, coordination and execution layers as User Management, Workflow Management, and Service Management layers respectively in the context of SBS. Multiple feedback control schemes are then developed for differentiated QoS requirements and resource constraints at each layer for adaptation. This provides a flexible solution to $Q1$ which allow various control theoretical approaches to be adopted to tackle specific control problems individually at each layer.

Our architecture is designed to address $Q2$ as well. The environment and resource situation is monitored by individual monitoring modules. The situation context are updated online and provided to the controller on each architectural layer to predict potential violations of timing and resource constraints for certain workflows. When a violation is detected, the corresponding controllers collaborate in a hierarchical manner and take adaptation actions to guarantee system performance and dependability. This enables us to overcome such violations with the minimal cost at runtime.

4. An Example

In this section, we will present an example to illustrate our approach to design an ASBS.

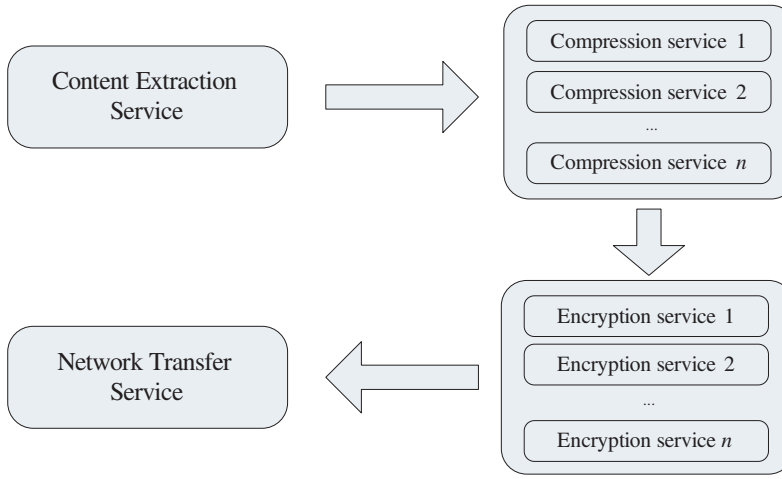

Fig. 3. The sequence workflow pattern of the example ASBS.

4.1. *Basic scenario*

Based on Hu and Jiang's previous work on adaptive server system [14], a multimedia web server providing media content compression, encryption and downloading services has been implemented as the test bed for deploying the example adaptive SBS.

There are two major QoS requirements for this example of ASBS: response time and security level. The server is expected to provide media content with acceptable delay and security level. Environment factors such as network congestion and security threats will affect the QoS. The example adaptive SBS is expected to handle such incidents properly. A sequential workflow pattern for a typical request handling process in the example adaptive SBS is depicted in Fig. 4. When user initiates a download request, the host compresses and encodes the request media content before sending it over the Internet to the user who later decompresses and decodes it.

4.2. *Control system structures*

In this subsection, the explicit design of the control system for each architectural layer is presented. Note that the control schemes presented here merely act as examples to demonstrate the feasibility of our architecture. The users can always develop their own control schemes for each layer based on their understanding of the architecture to satisfy specific various requirements.

4.2.1. *User management*

In this layer user requests are stored in a request pool, where they will be prioritized and put into the processing queue on the server side. Usually a traditional

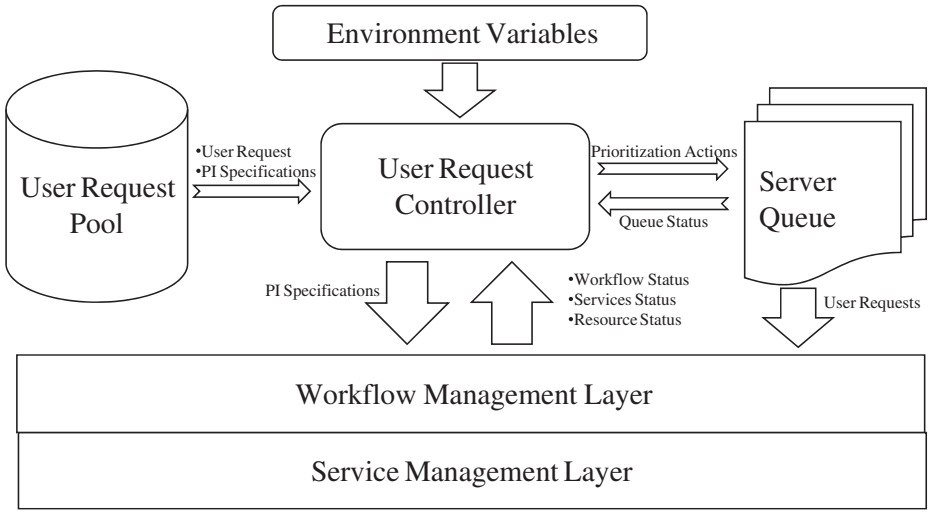

Fig. 4. User management control diagram.

SBS utilize a First-In First-Out (FIFO) policy to prioritize user requests. Other researchers have reported scheduling schemes which prioritize user requests according to a certain criteria, such as resource constraints or other QoS metrics. In this example we propose to use an adaptive control policy on the User Management layer to prioritize user requests in order to optimize the PI defined in this layer.

Figure 5 depicts block diagram of the User Control System: arriving user requests are stored in the User Request Pool together with PI specification tags; the User

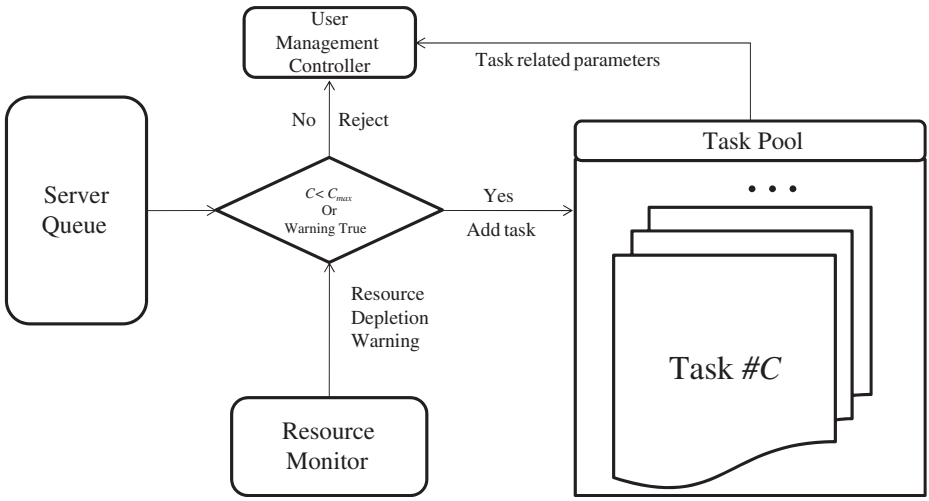

Fig. 5. Workflow management control diagram.

Request Controller applies an adaptive prioritization algorithm which generates actions to re-prioritize the server queue, which attempts to guarantee the overall PI of all requests in the queue; the server queue conducts prioritization actions and reports current queue status to the User Request Controller; Other parameters including workflow status, service status, resource status and environment variables are sent to the User Request Controller as feedbacks for decision making.

For each prioritization scheme the amount of unsatisfied requests is different and varies at runtime due to the fluctuation of network bandwidth and resource status. Moreover the total response time of each user request also varies from scheme to scheme. Thus the performance index here for user management is defined by the following formula:

$$PI_{\text{user}} = k_t \sum_{\text{queue}} T_{\text{res}} + k_v \sum_{\text{queue}} C_v \tag{2}$$

where T_{res} denotes the *Total Response Time* for each request in the current server queue, C_v is a binary variable which denote the estimation of possible PI violation for each request, k_t and k_v are weight factors respectively. T_{res} and C_v need to be estimated online by the User Management Controller using information on the workflow status, services status and resource status. More specifically, the request handling time for each request and the waiting time in the server pool are estimated and compared with user specification to see if there is a violation. The above definition of PI intends to address the tradeoff of user capacity and QoS performance.

Here is an example to explain how the adaptive controller works: when a user request arrives at the User Request Pool, it will immediately be sent to the User Request Controller together with its PI specifications. The User Request Controller will generate a number of prioritization scheme candidates for the server queue and the new user request. The PI_{user} of each candidate is calculated using environment variables and status feedbacks from the Server Queue, Workflow Management Layer and Service Management Layer. The candidate which gives the lowest PI_{user} will be selected to generate a prioritization action to re-prioritize the Server Queue. Since the parameters for PI_{user} calculation are updated on the fly, the controller is adaptive to changing environment and resource status.

4.2.2. *Workflow management*

When the controller in workflow management layer receives a task from user management layer, the workflow management selects a *workflow pattern* from a set of workflow patterns [35] according to the expected PI and current resource status. If no available workflow pattern can satisfy the expected PI, the workflow management layer will select the pattern that provides the best PI. According to the resource status and restriction, the controller will select the best pattern in the predefined workflow patterns to achieve optimal PI. For example, when a user requests several different contents that can be processed simultaneously, it might be better to select a parallel workflow pattern rather than a sequence pattern. Artificial intelligence

techniques, such as machine learning and genetic algorithms, can be applied to the workflow pattern selection.

In this paper, we designed and deployed a simple workflow management control policy as follows:

Step 1: Wait till the server queue forwards the next user request to the Workflow Management Layer.

Step 2: If $C = C_{\max}$, where C denotes the number of currently processing tasks and $C_{\max}$ is the threshold for simultaneous tasks on the server, send reject signal to User Management Controller and temporarily stop receiving new request from server queue. Goto *Step 1*.

If $C < C_{\max}$, spawn a new task to handle the request, set $C = C + 1$.

Step 3: If the Resource Monitor reports an expected resource depletion suspend the latest added task. Send Resource Depletion signal to User Management Controller and temporarily stop receiving new request from server queue.

Step 4: If a task has been accomplished, collect and send all task related parameters, such as encoding time or encrypting time, to the User Management Controller for estimating T_{res} and C_v. Remove the finished task and set $C = C - 1$.

Step 5: Goto *Step 1*.

Since the workflow management layer in this example does not involve the selection of pre-defined workflow patterns to satisfy various user requirements and resource constraints, the performance index at this layer is not specifically defined. The only adaptation action involved is when the maximal number of concurrent tasks is reached, the workflow management layer notify the user management layer to stop assigning new task and stop receiving requests from the server queue. The control flow is fairly simple so there is no need for defining a performance index as the control goal.

Figure 6 depicts the control diagram of the Workflow Management layer. Note that the workflow pattern used here is a self-expanding parallel workflow with resource and thread number constraints.

4.2.3. *Services management*

The controller in services management layer monitors the execution of the selected workflow pattern and its atomic services. When the status of service providers and environment variables changes, the controller initiates service adjustments to ensure the PI expectation can be met. Adjusting control actions include service provider selection and service parameter adjustment. For example, when a certain compression service is temporarily unreachable, the controller needs to find a substitute with the second best PI expectation. Since parameter estimation plays a heavy role in this process, an adaptive controller with quantitative parameter estimation module may be a good candidate here.

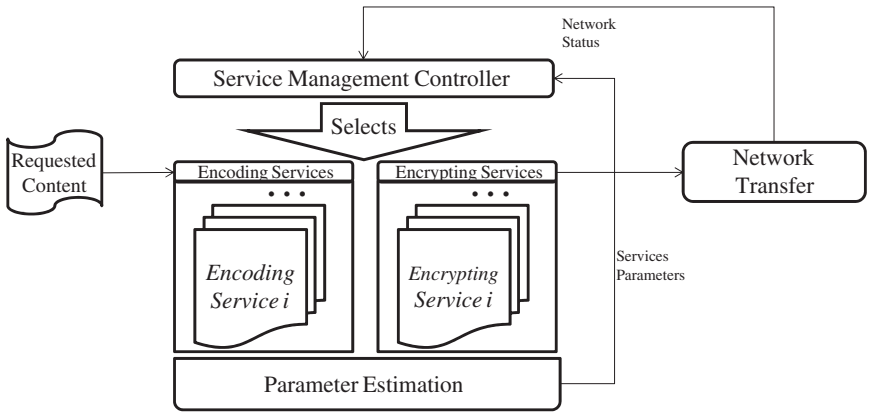

Fig. 6. Service management control diagram.

In this example, an adaptive Service Management Controller is designed and implemented to optimize performance index of each request handling task received from the Workflow Management Layer. According to the workflow pattern defined in the Workflow Management layer for this example, the performance index is defined as a function balancing total response time and actual security level for each user request handling task:

$$PI_{\text{service}} = k_t \sum \frac{1}{T_{\text{res}}} + k_s \sum S_{\text{actual}} \tag{3}$$

Note that one may argue the above definition might seem over simplified to address multiple QoS requirements. However the purpose of this example is to demonstrate the feasibility and the design process using our architecture rather than developing more complicated QoS models. The readers and practitioners are free to introduce their own QoS models into the designing of the controllers at each layer and will not affect the overall system structure. Adjustment might be needed for the controllers to monitor and control a customized PI but we won't go into it in this paper.

The Service Management Controller generates a number of combinations of encoding services and encrypting services, e.g. *Deflate algorithm* for encoding and 128 bit *DES* for encrypting. Then the PI_{service} of each combination is estimated using parameters collected from the parameter estimation module and network status feedback. The combination that provides the largest performance index will be used in the actual task. Figure 7 gives the control diagram of the Service Management Layer.

5. Execution of Experiments and Data Analysis

In order to demonstrate the feasibility of the three-layer architecture, a series of experiments has been conducted to examine the QoS performance of the example

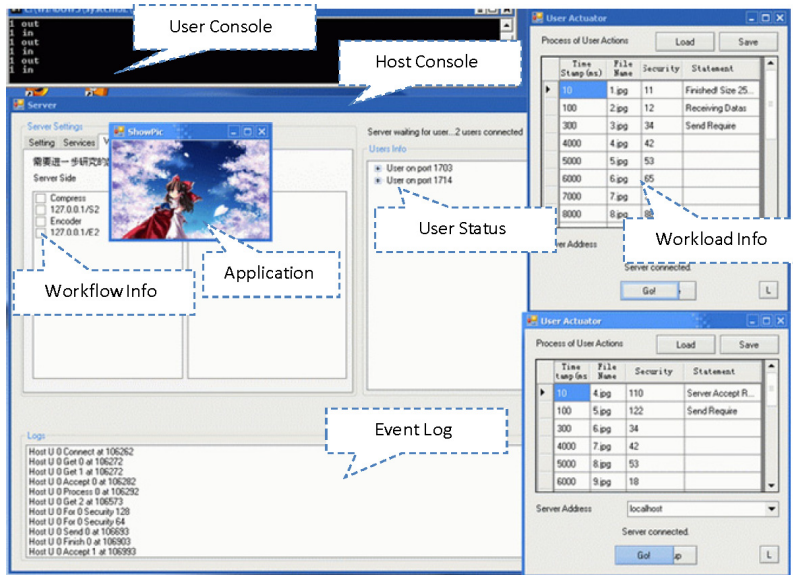

Fig. 7. The execution environment and components in the example of ASBS.

ASBS under various situations, such as facing an unstable network connection or random attack attempts at runtime.

The system is deployed on a distributed server group consisting of three servers: a host and two service providers running services implemented with C# and .Net framework. The servers are equipped with Intel Xeon 3.06 GHz CPU and 2.0 GB RAM, running Microsoft Window Server 2008 Standard. Four simulated users on four different PCs send requests to the host at random time. The each simulated user requests one of the four different workloads: hypertext files (.html), image files (.jpg), media files (.mp3) and miscellaneous which mixed with the aforementioned three types of files. There is also a simulated attack that initiates attack attempts to the host from time to time. We assume that all these attacks are blocked, but will raise the alert level of the host. The alert level determines the minimum required security level for encryption services. All users are required to send and receive their request with encryption key no shorter than the minimum level. There are three different encoding services on the encoding service provider and four encrypting services provided by the encryption service provider. More specifically, the three encoding services includes: *G-zip*, *Deflate* and *RAR3.* The four encrypting services includes: *DSA*, *DES*, *Triple DES* (*TDES*) and *Rijndael.* Each service locates on an individual host and announces itself to the service management controller in the service management layer at the beginning of the experiments. All encryption services can use encrypting key length ranging from 1 to 512 to provide various security levels for communications. Note that some encryption algorithms such as DES allow only a few fixed key lengths thus the flexibility of adaptation might be

reduced. A detailed analysis on the selection of key lengths for different encryption algorithms has lately been reported by the authors in [24]. The response time and the security level over a period of time were recorded.

The four workloads are generated as follows representing different type of users:

W_1: 100 web pages files, sizing from 6 KB to 170 KB, the average file size is 40 KB.
W_2: 100 JEPG picture files, sizing from 130 KB to 450 KB, the average file size 373 KB.
W_3: 20 MP3 music files, sizing from 0.7 MB to 11 MB, the average file size 4.4 MB.
W_4: 20 files randomly selected from the above three workloads, sizing from 23 KB to 9.9 MB, the average file size is 1.2 MB.

For each user request, we assign a predefined "deadline" which depends on the size of the content and the bandwidth of the user's connection. Basically the "deadline" is proportional to the file size in most cases. The security requirement is assigned arbitrarily for each request. Although the predefined user requirements seems coarse and might not represent actual user behavior, it provides a simple baseline to evaluate the system's QoS performance. Random noises are introduced to simulate the changing user requirement over runtime. More authentic simulation of user requirements will be considered in the future.

Figure 7 is a snapshot of the running system. Figures 8–11 depict the actual response time and security level (encryption key length) achieved with different number of concurrent users. Figure 8 depicts the actual response time (milliseconds) achieved for the single user (*user 4*) scenario. The red dashed line denotes the tentative "deadline" defined by the user for each request whereas the blue line denotes the actual response time. It is obvious that the actual response times never

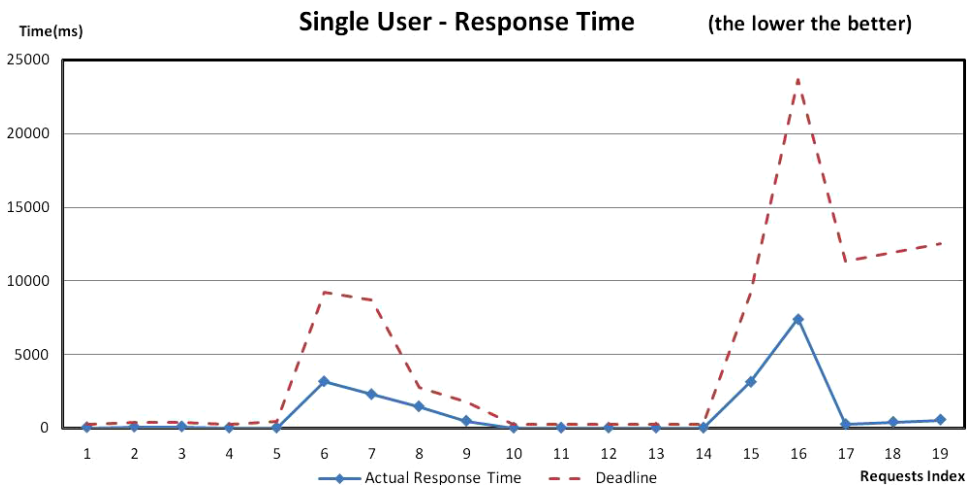

Fig. 8. The actual response time of the ASBS in single user scenario.

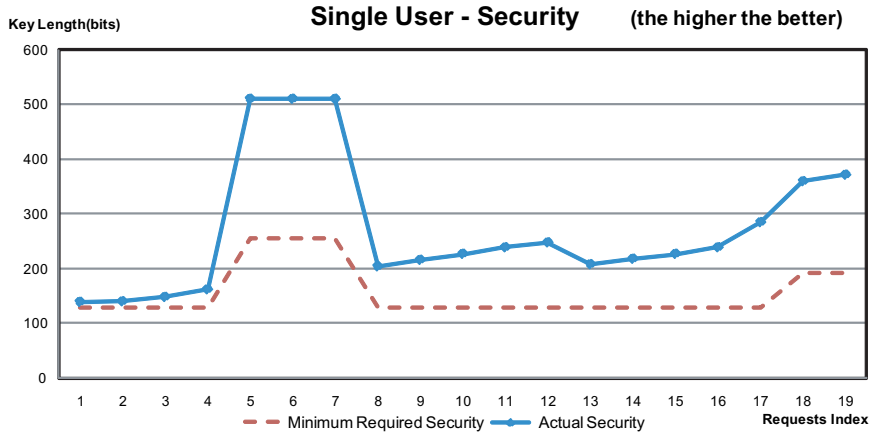

Fig. 9. The actual security level of the ASBS in single user scenario.

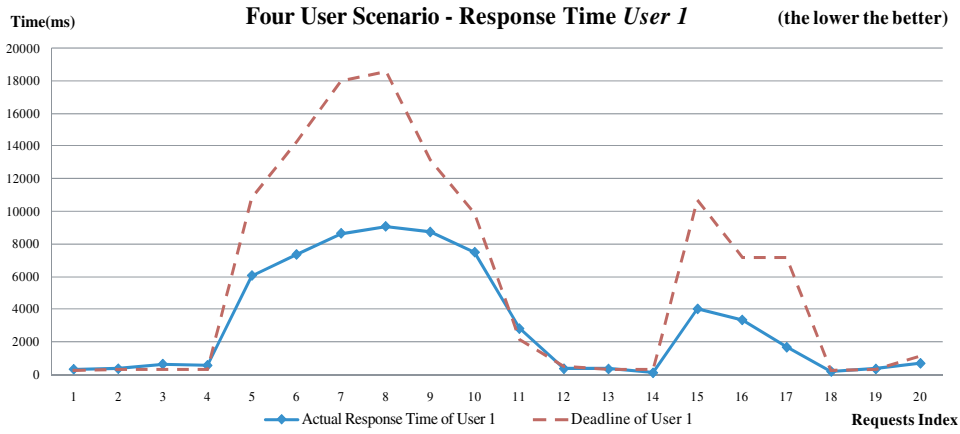

Fig. 10. The actual response time of *User 1* in the four user scenario.

exceed the deadlines, that is, all request are handled timely. There is an abrupt rise of response time which is because there is a simulated attack that caused the security level to be raised at the sixth request. Thus additional time is consumed for encryption with a higher level of key length. The size of the requested content also rises so the expected deadline is loosened and allowed a more robust encryption key to be used for this request. By selecting the appropriate encryption service, the extract time consumption for encryption did not exceed the deadline so a potential security violation is avoided in this case.

Figure 9 depicts the security performance of the ASBS in the single user scenario. The red dashed line denotes the minimum required security level and the blue line denotes the actual security level used for encryption for the user. Data showed that

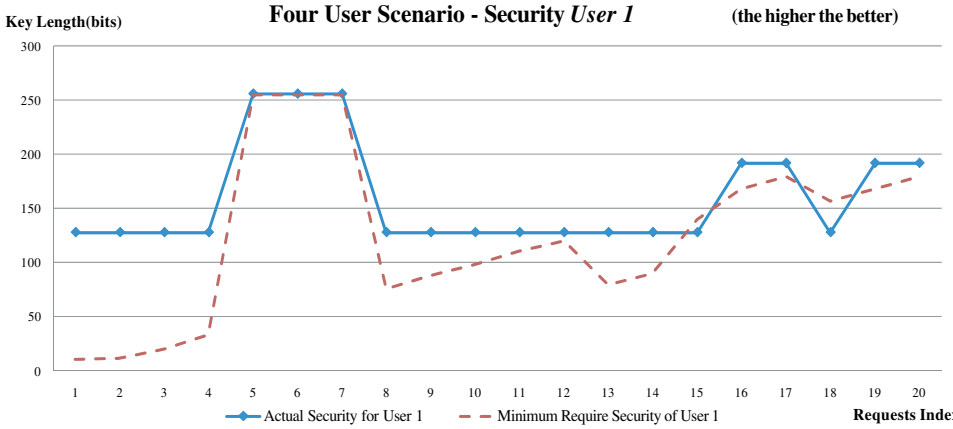

Fig. 11. The actual security level of *User 1* in the four user scenario.

no violation of security requirement happened during the session which is expected since it is a "light-load" scenario for the ASBS and it should be easy to guarantee multiple QoS requirements.

Figures 10 and 11 depicts the actual response time and security level achieve for *User 1* in the scenario with four concurrent users. Since there are multiple requests at the same time, the ASBS is expected to prioritize the requests and adaptively select the optimal workflow for each request. Furthermore the Service Management Controller should decide the optimal encoding and encryption services for each workflow. All control policies are designed to avoid violation of response time deadline and minimum required security level. Data shows that except for two missed deadlines at request #3 and #11, the response time performance of *User 1* is satisfactory for most of the requests. The two deadlines are missed merely by less than 100 milliseconds. The security performance is also desirable for there are only two requests that did not meet the minimum security requirement and the differences were insignificant. The ASBS demonstrate the advantage of its three-layer intelligent control structure by stabilizing multiple QoS performances in the multi-user and constantly changing environment. The robustness of the ASBS is also verified by data collected by the other three users: Figures 12–17 depict the response time and security level achieve by *User 2*, *User 3*, and *User 4*, respectively. Similar observations can be made from these figures.

6. Conclusions

How to design service based systems that are adaptable to dynamic user requirements, performance specifications and resource constraints is an important issue in distributed computing area. This paper investigates this issue from a software cybernetics perspective, and formulates the problem as a design problem of a complex

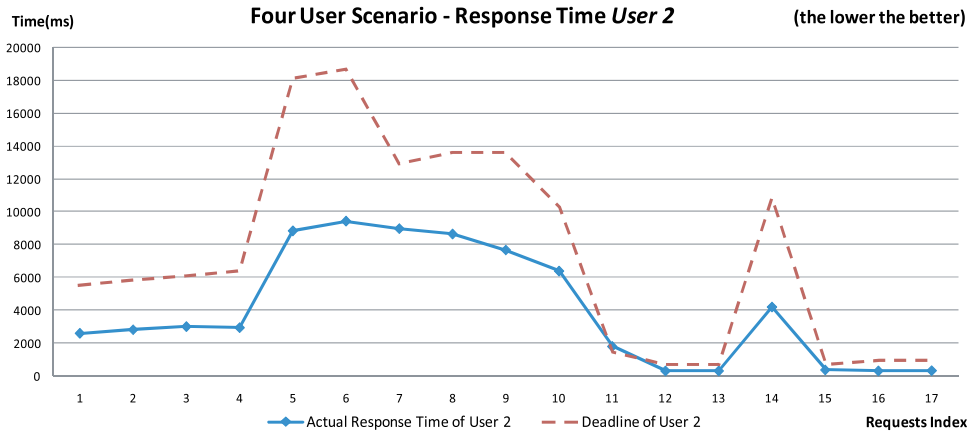

Fig. 12. The actual response time of *User 2*.

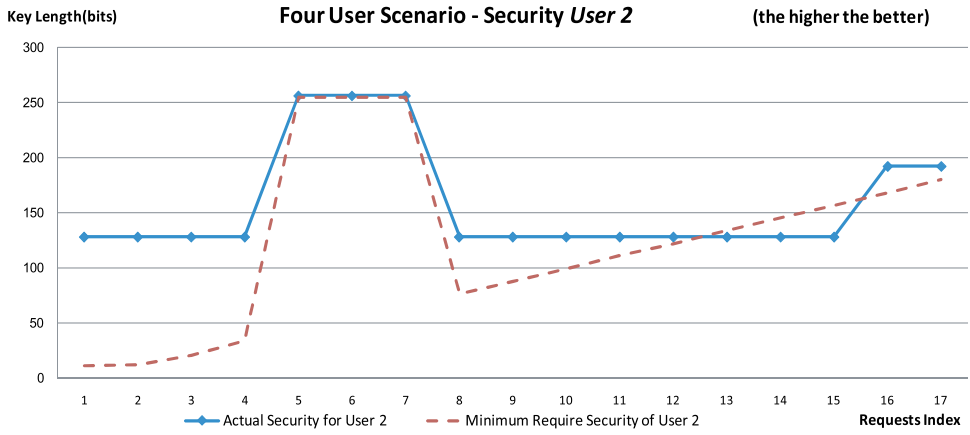

Fig. 13. The actual security level of *User 2*.

control system. Inspired by the underlying principles of the hierarchical intelligent control theory, we have presented a three-layer intelligent control architecture to the design and development of ASBS. The three architectural layers are User Management, Workflow Management, and Service Management respectively. By using different control schemes, each layer addresses a different aspect of the QoS and resource management issues in ASBS. The interactions among different layers are also discussed. Moreover, an example is provided to explain how to follow our architecture to design a simple ASBS. Experiments on the example ASBS with simulated users are conducted and the resulting data indicate that our architecture is feasible and can satisfy most of the user requirements in an open dynamic environment with a few exceptions.

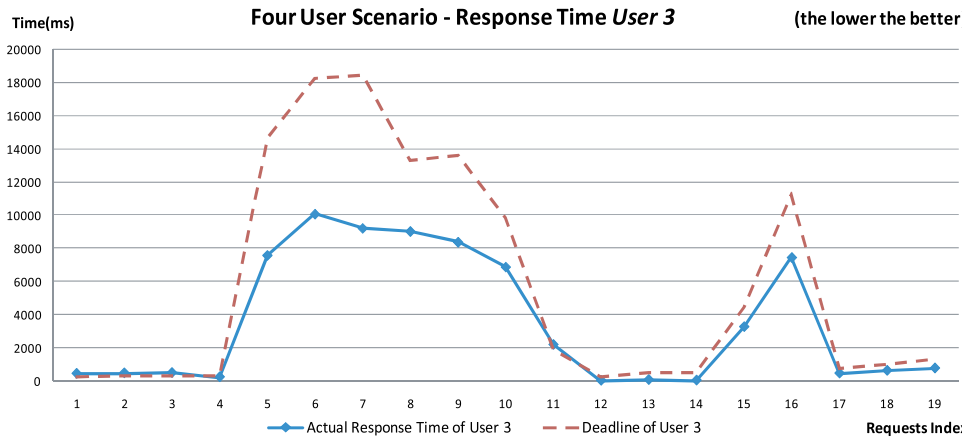

Fig. 14. The actual response time of *User 3*.

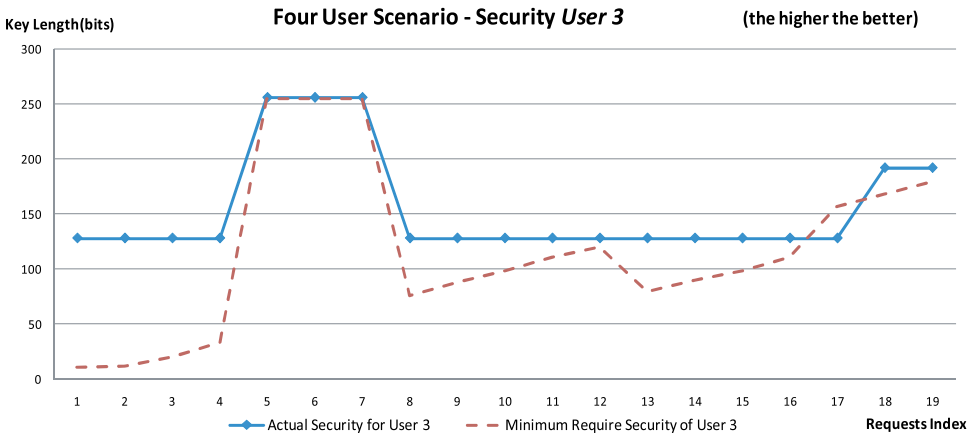

Fig. 15. The actual security level of *User 3*.

To sum up, our three-layer intelligent control architecture can properly address the research questions of this paper: (1) how to design an ASBS that is adaptable to changing user requirements, performance specifications and resource constraints and (2) how to detect or predict violations of timing and resource constraints for certain workflows in runtime, and adapt the system to overcome such violations. On the other hand, this study is also a contribution to the emerging area of software cybernetics that explores the interplay between software and control.

Future research might include the following topics: (1) examine and select better control algorithms and performance index definitions; (2) explore the QoS characteristics of different workflow patterns, and (3) automatically generate or synthesize the controllers on each layer of the adaptive SBS; (4) study how to balance the

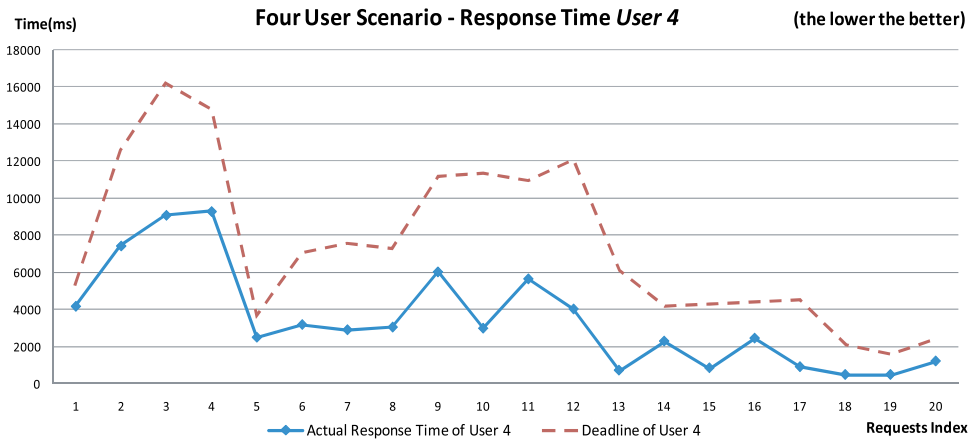

Fig. 16. The actual response time of *User 4*.

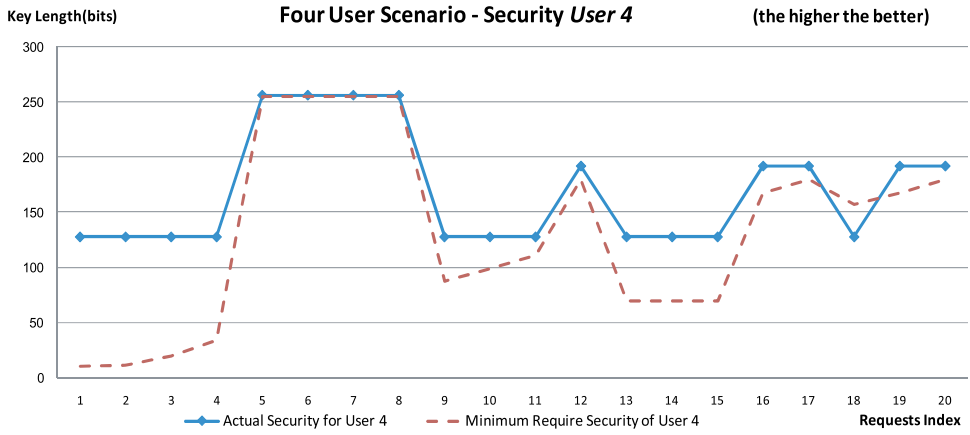

Fig. 17. The actual security level of *User 4*.

trade-off between efficiency and overhead of applying advanced control techniques; (5) apply this approach to more complicated applications.

References

1. T. F. Abdelzaher, K. G. Shin, and N. Bhatti, Performance guarantees for web server end-systems: A control-theoretical approach, *IEEE Trans. on Parallel and Distributed Systems* **13**(1) (2002) 80–96.
2. M. Aksit and L. Bergmans, Composing multiple-client-multiple-server synchronizations, in *Proc. IEEE Joint Workshop on Parallel and Distributed Real-Time Systems*, April 1997, pp. 269–282.
3. A. Arsanjani, Service-oriented Modeling and Architecture, http://www.ibm.com/developerworks/library/ws-soa-design1/, November 2004.

4. F. Bacchus and F. Kabanza, Using temporal logic to control search in a forward chaining planner, *New Directions in Planning*, eds. M. Ghallab and A. Milani (IOS Press, 1996), pp. 141–153.
5. BEA Systems, BEA SOA Domain Model White paper, http://www.ebizq.net/white_papers/6196.html.
6. K. Y. Cai, Optimal software testing and adaptive software testing in the context of software cybernetics, *Information & Software Technology* **44**(14) (2002) 841–855.
7. K. Y. Cai, J. W. Cangussu, R. A. Decarlo, and A. P. Mathur, An overview of software cybernetics, in *Proc. 11th Annual International Workshop on Software Technology and Engineering Practice*, September 2003, pp. 77–86.
8. G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, An approach for QoS-aware service composition based on genetic algorithms, in *Proc. 2005 Conference on Genetic and Evolutionary Computation*, June 2005, pp. 1069–1075.
9. S. W. Cheng, D. Garlan, B. Schmerl, P. Steenkiste, and N. Hu, Software architecture-based adaption for grid computing, in *Proc. 11th IEEE International Symposium on High Performance Distributed Computing 2002 (HPDC)*, July 2002, pp. 389–398.
10. X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao, AUTONOMIA: An autonomic computing environment, in *Proc. IEEE International Conference on Performance, Computing, and Communications*, April 2003, pp. 61–68.
11. D. F. Garcia, J. Garcia, J. Entrialgo, M. Garcia, P. Valledor, R. Garcia, and A. M. Campos, A QoS mechanism to provide service differentiation and overload protection to internet scalable server, *IEEE Trans. on Services Computing* **2**(1) (2009) 3–16.
12. C. Guo, M. Cai, and H. Chen, QoS-aware service composition based on tree-coded genetic algorithm, in *Proc. 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, July 2007, pp. 361–367.
13. R. B. Halima, K. Drira, and M. Jmaiel, A QoS-oriented reconfigurable middleware for self-healing web services, in *Proc. IEEE International Conference on Web Services 2008 (ICWS)*, September 2008, pp. 104–111.
14. H. Hu, C. H. Jiang, K. Y. Cai, and W. E. Wong, A control-theoretic approach to QoS adaptation in data stream management systems design, in *Proc. 4th IEEE International Workshop on Software Cybernetics (IWSC)*, July 2007, pp. 237–248.
15. G. Hwang, J. Shin, and J. W. Kin, Network-adaptive QoS control for relative service differentiation-aware video streaming, *Springer Berlin/Heidelberg, Management of Multimedia Networks and Services* **3754** (2005) 326–337.
16. M. W. Jang and G. Agha, Dynamic agent allocation for large-scale multi-agent applications, in *Proc. 1st International Workshop on Massively Multi-Agent Systems*, December 2004, pp. 19–33.
17. C. H. Jiang, H. Hu, K. Y. Cai, D. Huang, and S. S. Yau, An intelligent control architecture for adaptive service-based software systems with workflow patterns, in *Proc. 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC'08)*, July 2008, pp. 824–829.
18. J. Jin and K. Nahrstedt, On exploring performance optimization in web service composition, in *Proc. ACM/IFIP/USENIX International Middleware Conference*, October 2004, pp. 115–134.
19. Z. Jin and H. Zhu, A framework for agent-based service-oriented modeling, in *IEEE International Symposium on Service-Oriented System Engineering (SOSE) 2008*, December 2008, pp. 160–165.
20. N. Kandasamy, S. Abdelwahed, and J. P. Hayes, Self-optimization in computer systems via on-line control: Application to power management, in *Proc. 1st International Conference on Autonomic Computing*, May 2004, pp. 54–61.

21. P. Kruchten, H. Obbink, and J. Stafford, The past, present and future of software architecture, *IEEE Software* **23**(2) (2002) 22–30.
22. J. E. Lentz and L. Acar, A hierarchical intelligent controller for a three-link robot arm, in *Proc. 1995 IEEE International Symposium on Intelligent Control*, August 1995, pp. 209–214.
23. Y. Li, K. Sun, J. Qiu, and Y. Chen, Self-reconfiguration of service-based systems: A case study for service level agreements and resource optimization, in *Proc. IEEE International Conference on Web Services 2005 (ICWS)*, July 2005, pp. 266–273.
24. C. Liu, C. H. Jiang, H. Hu, K. Y. Cai, D. Huang, and S. S. Yau, A control-based approach to balance services performance and security for adaptive service based systems, to appear the *6th IEEE International Workshop on Software Cybernetics (IWSC)*, July 2009.
25. L. Ljung, *System Identification: Theory for the User, Second Edition* (Prentice Hall, 1998).
26. C. Y. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, Feedback control architecture and design methodology for service delay guarantees in web servers, *IEEE Trans. on Parallel and Distributed Systems* **17**(9) (2006) 1014–1027.
27. Y. Lu, A. Saxena, and T. F. Abdelzaher, Differentiated caching services: A control-theoretical approach, in *Proc. International Conference on Distributed Computing System*, April 2001, pp. 615–622.
28. Microsoft, Learn About Service Oriented Architecture (SOA), http://www.microsoft.com/biztalk/solutions/soa/overview.mspx, December 2006.
29. A. Mohammad, A. Chen, G. Wang, and C. Wang, A multi-layer security enabled quality of service (QoS) management architecture, in *Proc. 11th IEEE International Enterprise Distributed Object Computing Conference*, October 2007, pp. 423–423.
30. Oracle, Oracle Service-Oriented Architecture, http://www.oracle.com/technologies/soa/index.html.
31. C. Patel, K. Supekar, and Y. Lee, A QoS oriented framework for adaptive management of web service based workflows, *Location- and Context-Awareness*, LNCS 3479, pp. 16–25.
32. S. Ponnekanti and A. Fox, Sword: A developer toolkit for web service composition, in *Proc. 11th International World Wide Web Conference (WWW 2002)*, May 2002.
33. C. Raibulet, F. Arceil, S. Mussino, M. Riva, F. Tisato, and L. Ubezio, Components in an adaptive and QoS-based architecture, in *ACM Proc. 2006 International Workshop on Self-Adaption and Self-managing System*, May 2006, pp. 65–71.
34. J. Rao, P. Kungas, and M. Matskin, Application of linear logic to web service composition, in *Proc. 1st International Conference on Web Services (ICWS'03)*, June 2003, pp. 3–9.
35. N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar, Workflow control-flow patterns: A revised view, *BPM Center Report BPM-06-22*, BPMcenter.org, 2006.
36. G. N. Sardis, Architecture for intelligent machines, *Tech. Report CIRSSE-58*, RPI, 1991.
37. G. N. Saridis and M. C. Moed, Analytic formulation of intellingent machines as neural nets, in *Proc. IEEE International Symposium on Intelligent Control*, August 1988, pp. 22–27.
38. G. N. Saridis and H. E. Stephanou, A hierarchical approach to the control of a prosthetic arm, *IEEE Transaction on Systems, Man and Cybernetics* **7**(6) (1977) 407–420.
39. G. N. Sardis and K. P. Valananis, Analytic design of intelligent machines, *Automatica* **24** (1988) 123–133.

40. T. Shibata and T. Fukuda, Hierarchical intelligent control for robotic motion, *IEEE Trans. on Neural Networks* **5**(5) (1994) 823–832.
41. J. Siljee, S. Viintges, and J. Nijhuis, A context architecture for service-centric systems, *Database and Expert Systems Applications*, LNCS 2736 (Springer 2003), pp. 826–835.
42. E. Sirin, J. A. Hendler, and B. Parsia, Semi-automatic composition of web services using semantic descriptions, in *Proc. Web Services: Modeling, Architecture and Infrastructure (WSMAI) Workshop in Conjunction with the 5th International Conference on Enterprise Information Systems (ICEIS 2003)*, April 2003, pp. 17–24.
43. M. Stal, Using architectural patterns and blueprints for service-oriented architecture, *IEEE Software*, **23**(2) (2002) 54–61.
44. L. Tang, J. Dong, and T. Peng, A generic model of enterprise service-oriented architecture, in *IEEE International Symposium on Service-Oriented System Engineering (SOSE) 2008*, December 2008, pp. 1–7.
45. M. N. Varvatsoulakis, G. N. Saridis, and P. N. Paraskevopoulos, *IEEE Transactions on Robotics and Automation* (2000) 180–189.
46. W3C Web Services Architecture, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.
47. G. Wang, A. Chen, C. Wang, C. Fung, and S. Uczekaj, Integrated quality of service (QoS) management in service-oriented enterprise architecture, in *Proc. 8th IEEE International Enterprise Distributed Object Computing Conference 2004 (EDOC)*, 2004, pp. 21–32.
48. J. Wang, H. Chen, Y. Xu, and S. Liu, An architecture of agent-based intelligent control systems, in *Proc. 3rd World Congress on Intelligent Control and Automation*, July 2000, pp. 404–407.
49. S. J. Woodman, D. J. Palmer, S. K. Shrivastava, and S. M. Wheater, Notations for the specification and verification of composite web services, in *Proc. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC'04)*, September 2004, pp. 35–46.
50. K. Wu, D. J. Lilja, and H. Bai, An adaptive dual control framework for QoS design, *Springer Cluster Computing* **10**(2) (2007) 217–228.
51. S. S. Yau, H. Davulcu, S. Mukhopadhyay, D. Huang, Y. Yao, and H. Gong, Adaptable situation-aware secure service-based (AS3) systems, *Information Security Research*, eds. C. Wang and S. King, 2007, Chapter 5, pp. 585–596.
52. S. S. Yau, D. Huang, and L. Zhu, An approach to adaptive distributed execution monitoring for workflows in service-based systems, in *Proc. 4th IEEE International Workshop on Software Cybernetics (IWSC)*, July 2007, pp. 211–216.
53. S. S. Yau, D. Huang, L. Zhu, and K. Y. Cai, A software cybernetic approach to deploying and scheduling workflow applications in service-based systems, in *Proc. 11th International Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, March 2007, pp. 149–156.
54. S. S. Yau, N. Ye, H. Sarjoughian, and D. Huang, Developing service-based systems with QoS monitoring and adaptation, in *Proc. 12th IEEE Int'l Workshop on Future Trends of Distributed Computing Systems (FTDCS'08)*, October 2008, pp. 74–80.
55. A. Youssef, H. Abdel-Wahab, and K. Maly, The software architecture of a distributed quality of session control layer, in *Proc. 7th International Symposium on High Performance Distributed Computing 1998*, July 1998, pp. 21–28.