Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

# Adaptive service configuration approach for quality of service management in ubiquitous computing environments*

Yong ZHANG†, Shen-sheng ZHANG, Song-qiao HAN

(*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*)

†E-mail: zycs921@gmail.com

Received Aug. 1, 2008; Revision accepted Sept. 30, 2008; Crosschecked Apr. 27, 2009

**Abstract:** Mobility and resource-limitedness pose challenging issues to service configuration for quality of service (QoS) management in ubiquitous computing environments. Previous configuration approaches, such as static resource reservation, dynamic resource allocation and single service composition are not valid in the environments. In this study, we present an adaptive service configuration approach. Firstly, we reduce the dynamic configuration process to a control model which aims to achieve the variation of critical QoS on minimal level with less resource cost. Secondly, to deal with different QoS variations, we design two configuration strategies—service chain reconfiguration and QoS parameter adjustment—and implement them based on fuzzy logic control theory. Finally, a configuration algorithm is developed to flexibly employ the two configuration strategies in tune with the error of critical QoS in configuration process. The results of simulation experiments suggest that our approach outperforms existing configuration approaches in both QoS improvement and resource utilization.

**Key words:** Service configuration, Ubiquitous computing (UbiComp), Quality of service (QoS), Fuzzy logic control
**doi:**10.1631/jzus.A0820422 **Document code:** A **CLC number:** TP31

## INTRODUCTION

With the rapid development of wireless communication and micro-electronics technology, ubiquitous computing (UbiComp), which is built on wireless network and portable devices, has aroused widespread interest in academic and industrial communities. Recently, many research projects have been launched to explore and develop a variety of UbiComp applications with smart phones (Roussos *et al*., 2005), such as the healthcare monitoring and alerting service E-Care (Marsh, 2002), the exhibition navigation application mXpress (Giaglis *et al*., 2002), and the pervasive retail system MyGrocer (Kourouthanassis and Roussos, 2003). However, mobility and resource-limitedness pose challenging issues to service configuration for QoS management in UbiComp environments. Imagine a wireless video monitor application: at the mobile-end a user captures live video images with the built-in camera of the portable device, and the captured images are preprocessed (e.g., compressed or encrypted) and transmitted over wireless network to the server-end for real-time monitoring. The crucial issue for service configuration is how to enable the application to offer satisfactory critical QoS (e.g., the frame rate of video images) with limited memory and CPU power in a bandwidth-fluctuating environment.

Service configuration for QoS management has attracted many researchers for a decade. Li and Nahrstedt (1999) presented a proportional-integral-derivative (PID) control algorithm and fuzzy control model that could allocate CPU and bandwidth resource dynamically to improve track precision. Xu *et al*.(2000) and Gu and Nahrstedt (2002) employed the compositions of service components to meet the QoS requirements in multimedia applications. Huang *et al*.(2004) developed resource-reservation schemas

* Project (No. 05SN07114) supported by the International Cooperation Project of the Shanghai Science and Technology Commission of China and the National Research Council of Canada

to guarantee QoS for the real-time multimedia services. However, these configuration methods, including static resource reservation and dynamic resource allocation, are not valid in resource-limited UbiComp environments. Moreover, current wireless techniques cannot completely hide all mobile link effects. Resource-limitedness and variations in link quality inevitably cause QoS variations. A single service composition method cannot improve QoS effectively. Hence, we have tried to develop a novel service configuration approach which specifies a range of acceptable QoS levels rather than guarantee the specified QoS value. Effective resource utilization is also a necessary design consideration for the configuration approach in resource-limited environments.

In this study, we propose an adaptive service configuration approach for QoS management in UbiComp environments. The dynamic configuration process is reduced to a control model which aims to achieve critical QoS variations on minimum level with less resource cost. Inspired by Chan and Chuang (2003), we design QoS parameter adjustment and service chain reconfiguration strategies to handle different QoS variations. The two configuration strategies are implemented with two fuzzy logic controllers (FLCs) to deal with non-linear configuration process. Furthermore, we develop a configuration algorithm that can employ the strategies flexibly to improve critical QoS in tune with the error of critical QoS.

## SERVICE MODEL

### Service component

According to World Wide Web Consortium (2007), a service component is an accessible software entity that can be discovered and invoked by other components or applications. A service component consumes system resources, performs certain functions and delivers results to other service components and end users. We adopt this meaning and focus on the properties associated with the QoS of service components. In particular, we give a formal definition as follows:

**Definition 1** (Service component) A service component can be denoted as $S=\{V_s, I_s, O_s, Q_s, P_s\}$, where $V_s$ is the service itself that can be described in terms of function, resource and security requirements, etc.; $I_s$ is a set of input variables, data formats and communication protocols, etc.; $O_s$ is a collection of output results, data formats and communication protocols, etc.; $Q_s$ represents a collection of QoS attributes such as delay, CPU utilization, and memory cost; $P_s$ denoting QoS parameters, represents the collection of the component's parameters which can be adjusted to influence the values of the QoS attributes.

Fig.1 shows an example of a service component that can perform data compression. It should be noted that the compression ratio and block size are QoS parameters in the service component. A detailed example for QoS parameters will be presented in the next section. Generally, the description of a service component is encoded by web ontology language (OWL) within a service profile and published by service providers.

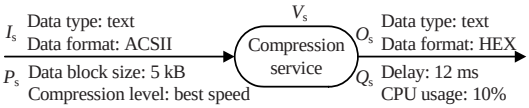


**Fig.1 Representation of a data compression service component**

### Composite service specification

A composite service or application is specified as a set of user tasks which are combined according to control-flow and data-flow dependencies, and implements the aggregated functions of the tasks (Zeng and Benatallah, 2004). We describe a composite service by a task graph which is defined as follows:

**Definition 2** (Task graph) A task graph is a directed acyclic graph, which can be represented as $G=\{U_G, R_G\}$, where $U_G=\{u_0, u_1, \ldots, u_n\}$ is a set of task nodes which contains the required tasks to accomplish the functions of the composite service; $R_G$ is a set of directed edges which represents the dependency relationships among the tasks; $u_0$ is the initial task only with outgoing edges, while $u_n$ is the final task only with incoming edges.

Fig.2 illustrates a simplified task graph for the wireless video monitor application. The task $u_0$ takes charge of capturing images; $u_1$ can transcode images by altering the color depth of images; $u_2$ is responsible for image encryption, and $u_3$ is used to transmit data over wireless network.

We introduce the concept of a service chain to describe the runtime instances of a composite service.

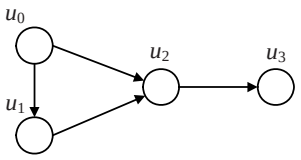


**Fig.2 Task graph of wireless video monitor application**

The definition of a service chain can be described as follows:

**Definition 3** (Service chain) A service chain for task graph $G$ is represented by $G^c$, a series of pairs ($u_i$, $s_i$) ($i$=0, 1, …, $n$) linked together, if:

$G_u^c = \{u_0, u_1, ..., u_n\}$ is a subset of $U_G$ in $G$. $G_s^c = \{s_0, s_1, ..., s_n\}$ is a set of required service components to accomplish the tasks in $G_u^c$;

$u_0$ is the initial task and $u_n$ is the final task. For each $u_i$ ($i \in [1, n-1]$), $u_{i-1}$ is its predecessor and $u_{i+1}$ is its successor in $G$;

For each pair ($u_i$, $s_i$) in $G^c$, the service component $s_i$ can accomplish the task $t_i$.

Generally, a composite service has multiple service chains. We assume there is only one service chain applied in a moment of runtime. In particular, we use $G^c(t)$ to denote the applied service chain at time $t$ in runtime. The corresponding task set and service component set involved in $G^c(t)$ are denoted as $G_u^c(t)$ and $G_s^c(t)$, respectively. Fig.3 shows multiple service chains for the task graph of the wireless video monitor application. $s_0$ is an image capture component, $s_1$ is an image transcoder and $s_5$ is a network transmitter. $s_2$, $s_3$ and $s_4$ are alternative encryption service components. Suppose the service chain at time $t_0$ is $G^c(t_0)$: $(u_0, s_0)\rightarrow(u_2, s_2)\rightarrow(u_3, s_5)$ and at time $t_1$ is $G^c(t_1)$: $(u_0, s_0)\rightarrow(u_1, s_1)\rightarrow(u_2, s_4)\rightarrow(u_3, s_5)$, then $G_u^c(t_0) = \{u_0, u_2, u_3\}$, $G_u^c(t_1) = \{u_0, u_1, u_2, u_3\}$, $G_s^c(t_0) = \{s_0, s_2, s_5\}$ and $G_s^c(t_1) = \{s_0, s_1, s_4, s_5\}$.

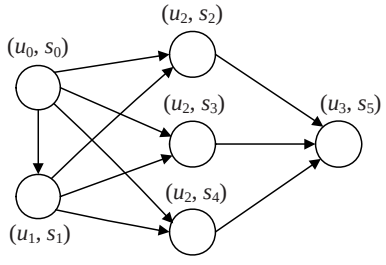


**Fig.3 Service chains for the task graph in Fig.2**

## OUR APPROACH

In this section, we firstly discuss QoS expressions for service components and composite service in runtime. Secondly, we model the configuration process and present two configuration strategies as well as their implementation based on fuzzy logic control theory. Finally, we propose a configuration algorithm to apply the two service strategies flexibly. For simplicity, we assume that an application has only one critical QoS attribute in this study.

### QoS expressions

As mentioned above, the QoS values of a service component are determined by the related QoS parameters. We use $q_\tau(s_i, t)$ to represent the value of the QoS attribute $q_\tau$ of service $s_i$ at time $t$. Then, $q_\tau(s_i, t)$ can be expressed as follows:

$$q_\tau(s_i, t) = g_\tau(p_0^t, p_1^t, ..., p_m^t), \quad (1)$$

where $p_j^t$ represents the value of QoS parameter $p_j$ at time $t$, $p_j \in P_{s_i}$, $j \in [0, m]$ and $g_\tau : p_0^t \times p_1^t \times ... \times p_m^t \rightarrow q_\tau(s_i, t)$ is a function mapping the QoS parameters to the QoS value. For example, we investigate the file compression service LightNzip (Toysoft, 2005) performing on a Palm Tungsten T2 with an ARM processor. The service's delay is determined by two QoS parameters—compression level and file size. Table 1 lists several mappings in the service. The compression level can be configured by the user with two modes: HiSpeed (high speed and low compression) and HiCompress (high compression and low speed).

**Table 1 Partial mappings in LightNzip**

| Compression level | Delay (ms) | | |
|---|---|---|---|
| | 10* | 20* | 50* |
| HiSpeed | 64 | 113 | 257 |
| HiCompress | 253 | 435 | 902 |

* File size, kB

In runtime, the QoS of a composite service or application is dependent on the QoS of all service components in its service chains. As for a composite service or application represented by task graph $G$, we use $q_\tau(G, t)$ to denote the value of QoS attribute $q_\tau$ at time $t$. So, $q_\tau(G, t)$ is denoted as follows:

$$q_\tau(G,t) = f_\tau(q_\tau(s_1,t), q_\tau(s_2,t), ..., q_\tau(s_n,t)), \quad (2)$$

where $s_i \in G_s(t)$, $i \in [0,n]$ and $f_\tau$ is a QoS attribute-specific function. For example, the memory cost of an application at time $t$ equals the total memory cost of all service components in its service chain:

$$q_{\text{memory}}(G,t) = \sum_{s_i \in G_s^c(t)} q_{\text{memory}}(s_i,t). \quad (3)$$

**Service configuration model**

The critical QoS variation is inevitable when available resource change occurs. The objective of service configuration is to manage critical QoS variation to implement critical QoS adaptation and effective resource utilization. To achieve the objective, service configuration should improve the critical QoS with all available resource when the critical QoS drops below the user-specified target value. Conversely, it should pull down the critical QoS to save the resource when the critical QoS is beyond the target value. We measure the variation with critical QoS disturbance which is defined as follows:

**Definition 4** (Critical QoS disturbance) In runtime of an application denoted by task graph $G$, the disturbance of critical QoS $q_\tau$ is denoted as follows:

$$D_{du}(G) = \int_{t_0}^{t_1} | q_\tau(G,t) - q_0 | \, \mathrm{d}t, \quad (4)$$

where $q_0$ represents the user-specified target value of $q_\tau$, $q_\tau(G, t)$ is the value of $q_\tau$ at time $t$, $|q_\tau(G, t)-q_0|$ denotes absolute deviation of $q_\tau$ from $q_0$ at time $t$, and $du=[t_0, t_1]$ is the duration of runtime.

In our approach, the configuration process can be reduced to a control model which tries to achieve minimal critical QoS disturbance $D_{du}(G)$ with less resource cost in runtime. Fig.4 illustrates the control model in which an application is considered as a controllable target system. A context manager is a situation monitor which can collect the application's output as the feedback for configuration iteration. Working as a controller, the service configurator receives the context information associated with the situation of critical QoS and resource cost from the context manager. It then applies specific configuration schema and dispatches configuration instructions to rule out the difference between the target value $q_0$ and the monitored critical QoS $q_\tau(G, t)$. To achieve flexibility in a control system, control operations imposed on the target system are usually conducted by an actuator rather than the controller itself. In our design, the service adapter acts as an actuator which receives configuration instructions from the service configurator and conducts concrete configuration operations on the application.

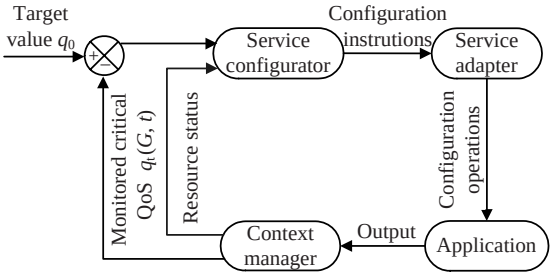


**Fig.4 Control model for service configuration**

**Service configuration strategies**

To deal with inconstant environments effectively, we designed two configuration strategies for the service configurator:

(1) QoS parameter adjustment: according to Eqs.(1) and (2), we know that the application's QoS is associated with the QoS of service components in the service chains, and the values of QoS parameters determine the QoS of the service components. Hence, we can adjust the application's QoS by tuning the values of QoS parameters in involved service components;

(2) Service chain reconfiguration: according to Eq.(2), the application's QoS is closely related to service chains. Hence, we can manipulate the application's QoS by adding, deleting as well as substituting involved service components in the service chains.

The two configuration strategies demonstrate different features. Compared with QoS parameter adjustment, service chain reconfiguration introduces more delay and system cost owing to application-level maintenance rather than simple interior parameter re-settings. Service chain reconfiguration can be considered as a coarse-grained strategy by which QoS can be adjusted sharply, but it tends to cause negative effects such as overshoot and oscillation. On the other hand, QoS parameter adjustment is a

fine-grained strategy by which QoS can be improved in a delicate manner but it needs more time to reach the target value. Examples of the two strategies will be presented in the next section.

**Fuzzy logic control implementation**

In mobile and resource-limited environments, dynamic service configuration that aims to control critical QoS disturbance on minimal level is complex and non-linear. In our approach, we design a service configurator by employing fuzzy logic control that is a well-known solution to non-linear systems (Su and Stepanenko, 1994; Tsay *et al*., 1999). Instead of using a complex mathematical model, fuzzy logic control uses linguistic descriptions to define the relationship between the input information and the output action. Moreover, the rule-base fuzzy inference can be easily updated and expanded to improve control performance.

Fig.5 illustrates a concrete architecture of service configuration. Within the service configurator, the two configuration strategies are implemented with service chain fuzzy logic controller (SC-FLC) and QoS parameter fuzzy logic controller (QoSP-FLC). Each FLC comprises four components: fuzzifier, fuzzy rule base, fuzzy inference engine and defuzzifier. The fuzzifier can map a crisp input to one or more fuzzy sets which are represented by membership functions defined on the universe of discourse. The universe of discourse is the space where the fuzzy variables are defined. The membership function gives the grade, or degree, of membership within the set. The fuzzy rule base is a set of fuzzy rules in the form of if-then statements:

Rule $i$: if $x_1$ is $A_{i1}$, $x_2$ is $A_{i2}$, …, $x_m$ is $A_{im}$, then $y$ is $B_i$,

where $i$=1, 2, …, $n$, $x_j$ ( $j$=1, 2, …, $m$) are input linguistic variables associated with the situation of critical QoS and resource cost, and $y$ represents the output linguistic variables associated with configuration instructions; $A_{ij}$ and $B_i$ are the corresponding linguistic values which are characterized by the membership functions $\mu_{A_{ij}}(x_j)$ and $\mu_{B_i}(y)$, respectively. Each rule represents a fuzzy implication: $A_{i1} \times A_{i2} \times ... \times A_{im} \rightarrow B_i$.

The fuzzy inference engine invokes each appropriate rule and generates a result for each, then combines the results of the rules. The defuzzifier converts the combined result back into a specific configuration instruction. Suppose that we adopt centroid method in the defuzzifier, the output of the FLC can be represented as follows:

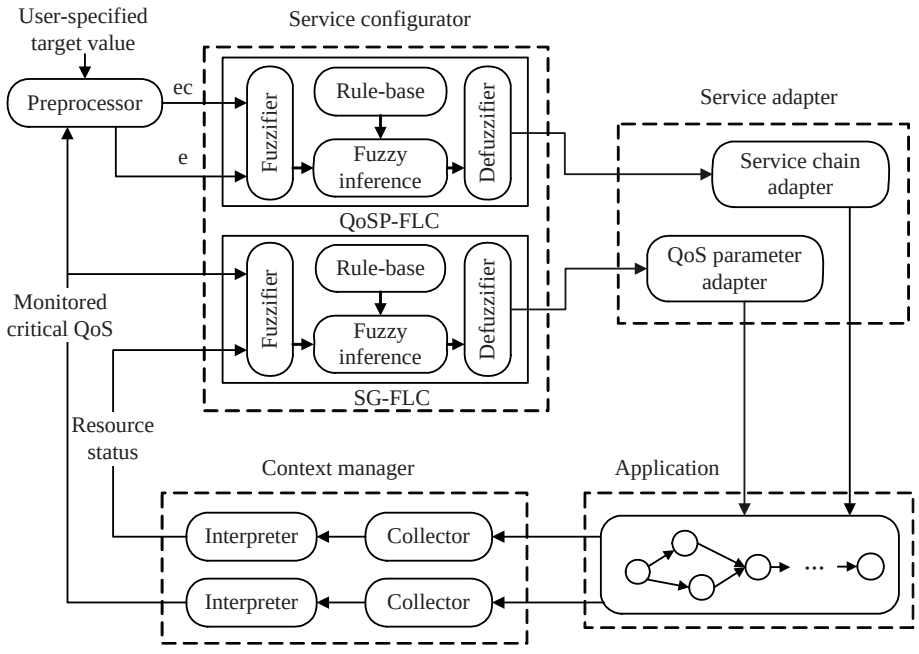


**Fig.5 Adaptive service configuration based on fuzzy logic control**
ec: error change; e: error

$$y=\frac{\sum_{i=1}^{n}\mu_{B_i}(y)\cdot\left(\sum_{j=1}^{m}\mu_{A_{ij}}(x_j)\right)}{\sum_{i=1}^{n}\left(\prod_{j=1}^{m}\mu_{A_{ij}}(x_j)\right)}. \quad (5)$$

In the configuration process, context interpreter is responsible for producing high-level context information based on low-level context information from the collectors on the portable device (Dey *et al*., 2001). The SC-FLC takes the error of critical QoS and the resource status of the portable device, and produces the instructions for service chain reconfiguration. The QoSP-FLC receives the error and the error change of critical QoS as its input, and dispatches the instructions for the adjustment of the involved components' QoS parameters. The corresponding service adapters are responsible for manipulating the service chain and re-setting the values of QoS parameters in the involved components.

However, the fluctuation of critical QoS introduced by a fast-changing environment creates a need for the service configurator to apply the two strategies flexibly in the configuration process. The SC-FLC is suited to handle the situation in which the critical QoS varies greatly, while the QoSP-FLC is suited to deal with a slight variation of critical QoS. Thus, we develop a configuration algorithm which enables the service configurator to apply the two FLCs dynamically depending on whether the error of critical QoS is in the range [$\lambda_0$, $\lambda_1$] or [$\lambda_2$, $\lambda_3$], where $\lambda_0$ and $\lambda_1$ are application-specific and represent the floor and ceiling of the errors of critical QoS for the QoSP-FLC, and $\lambda_2$ and $\lambda_3$ are those for the SC-FLC. Moreover, $\lambda_3>\lambda_1$ and $\lambda_2<\lambda_0$. The pseudocode of the algorithm is described as follows:

```
initSC_FLC(); initQoSP_FLC(); target QoS←q0; previousError←0; errorFlag←0; errorFloor0←λ0; errorCeiling0
←λ1; errorFloor1←λ2; errorCeiling1←λ3;
While (.True.)
  {contextQoS=getContextEvent(QoSAttributeID);
  currentResource=getContextEvent(resourceID);
  currentError=currentQoS−targetQoS;
  errorChange=currentError−previousError;
If (currentError in [errorFloor0, errorCeiling0])
  {errorFlag=1;}
else if
  (currentError in [errorFloor1, errorCeiling1])
  {errorFlag=2;}
  If (errorFlag≠2)
    {QoSP_FLC.setInput(currentResource, currentQoS);
    configurationInstruction=QoSP_FLC.infer();
    QoSParameterAdapter.execute(configurationInstruction);}
  If (errorFlag≠1)
    {SC_FLC.setInput(currentError, errorChange);
    configurationInstruction=SC_FLC.Infer();
    serviceChainAdapter.execute(configurationInstruction);
  previousError=currentError; errorFlag=0;}
}
```

## CASE EXAMPLE

To have a clearer understanding of the fuzzy logic control method, we will show how to design the FLCs in the service configurator. The wireless video monitor application is used as our example. Note that the images in the application are in JPEG format. As mentioned above, the frame rate of images is the critical QoS attribute. The mobility of users with portable devices may produce the fluctuation of bandwidth which seriously affects the frame rate of images. So, the service configurator aims to control the variation of frame rate on minimum level with less resource cost. According to the application's description stated previously, we identify configuration options as follows:

(1) Configuration for the service chain:

(a) Adding/dropping transcoder: In the same bandwidth environments, the smaller the data size of the image, the higher the frame rate. In the application, the data size of the image can be reduced by employing an image transcoder that can transfer full-colored images to 256-color images. So, adding or dropping an image transcoder can affect the frame rate of the image.

(b) Applying an appropriate encryption engine: Image encryption is a task that influences the frame rate. In the current implementation, the task is implemented with the Bouncy Castle (2005) crypto package which offers three alternative engines: AESLightEngine is an implementation optimized for low memory usage, AESFastEngine is optimized for speed, and AESEngine is the compromise of the previous two. Applying an appropriate encryption engine is a practical approach to improve frame rate and resource utilization.

(2) Configuration for QoS parameters: the data size of the image can also be altered by adjusting the image quality of the image capture component in

which the image quality is an integer with a scale of 0~100, and represents the compression level of the JPEG image captured through the camera (JCP, 2005; Forum Nokia, 2007). The trade-off is that the lower image quality means that the data size of the image is smaller but the frame rate is higher. So, the frame rate can be improved by tuning the image quality of the image capture component dynamically.

Compared with adjusting image quality, image transcoding and encryption are computing-intensive and resource-consuming tasks. Reconfiguring the service chain must take account of both the frame rate and the resource cost related to memory and CPU usage in the resource-limited portable device. Thus, *Frame_rate*, *CPU_usage*, and *Memory* are identified as input linguistic variables of the SC-FLC. The output of SC-FLC is represented in the form of service chain reconfiguration instructions that include adding or dropping an image transcoder and applying an appropriate encryption engine. On the other hand, adjusting image quality is a delicate approach to dealing with slight variation in the frame rate. Hence, *Frame_rate_error* and *Frame_rate_error_change* act as input linguistic variables of the QoSP-FLC. Naturally, *Image_quality_configuration* acts as the output linguistic variable. Table 2 lists all input and output linguistic variables in the two FLCs.

Tables 3 and 4 describe the rules in the SC-FLC and the QoSP-FLC, which represent the mapping from the situation of frame rate and resource cost to configuration instructions. The linguistic values in the rules are defined by triangular and trapezoidal shaped membership functions on the universe of discourses of corresponding linguistic variables.

Figs.6 and 7 show the membership functions. Usually, the membership functions are application-specific. In Table 4, Figs.6 and 7, NB, NM, NS, NO, O, PO, PS, PM, and PB are linguistic values representing negative big, negative medium, negative small, negative zero, zero, positive zero, positive small, positive medium, and positive big, respectively.

## EXPERIMENTS AND ANALYSIS

In this section, we firstly describe current implementation for simulation experiments. Then we briefly introduce the testbed, performance metrics and experimental scenarios, etc. Finally, we present the results of the simulation experiments.

**Table 4 Fuzzy rules in the QoSP-FLC**

| *Image_quality_configuration* | | *Frame_rate_error_change* | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | NB | NM | NS | NO | PO | PS | PM | PB |
| *Frame_rate_error* | NB | PB | PB | PM | PM | PS | PS | PS | O |
| | NS | PB | PM | PM | PM | PS | O | O | NS |
| | O | PM | PM | PS | O | O | NS | NM | NB |
| | PS | PM | PS | O | O | NS | NS | NM | NB |
| | PB | PS | O | O | NS | NM | NM | NB | NB |

**Table 2 Linguistic variables in the two FLCs**

| Controller | Linguistic variable | Type | Universe of discourse |
|---|---|---|---|
| SC-FLC | *Frame_rate* (frame/s) | Input | [0, 20] |
| | *Memory* (MB) | Input | [0, 5] |
| | *CPU_usage* (%) | Input | [0, 40] |
| | *Service_chain_configuration* | Output | [0, 1] |
| QoSP-FLC | *Frame_rate_error* (frame) | Input | [−4, 4] |
| | *Frame_rate_error_change* (frame) | Input | [−3, 3] |
| | *Image_quality_configuration* | Output | [0, 100] |

**Table 3 Fuzzy rules in the SC-FLC**

| |
|---|
| If (*Frame_rate* is Low) and (*Memory* is not Low), then *Service_chain_configuration* is Apply_AESFastEngine |
| If (*Frame_rate* is Low) and (*CPU_usage* is not Low), then *Service_chain_configuration* is Add_Transcoder |
| If (*Frame_rate* is Medium), then *Service_chain_configuration* is Apply_AESEngine |
| If (*Frame_rate* is High) and (*CPU_usage* is not High), then *Service_chain_configuration* is Drop_Transcoder |
| If (*Frame_rate* is High) and (*Memory* is not High), then *Service_chain_configuration* is Apply_AESLightEngine |

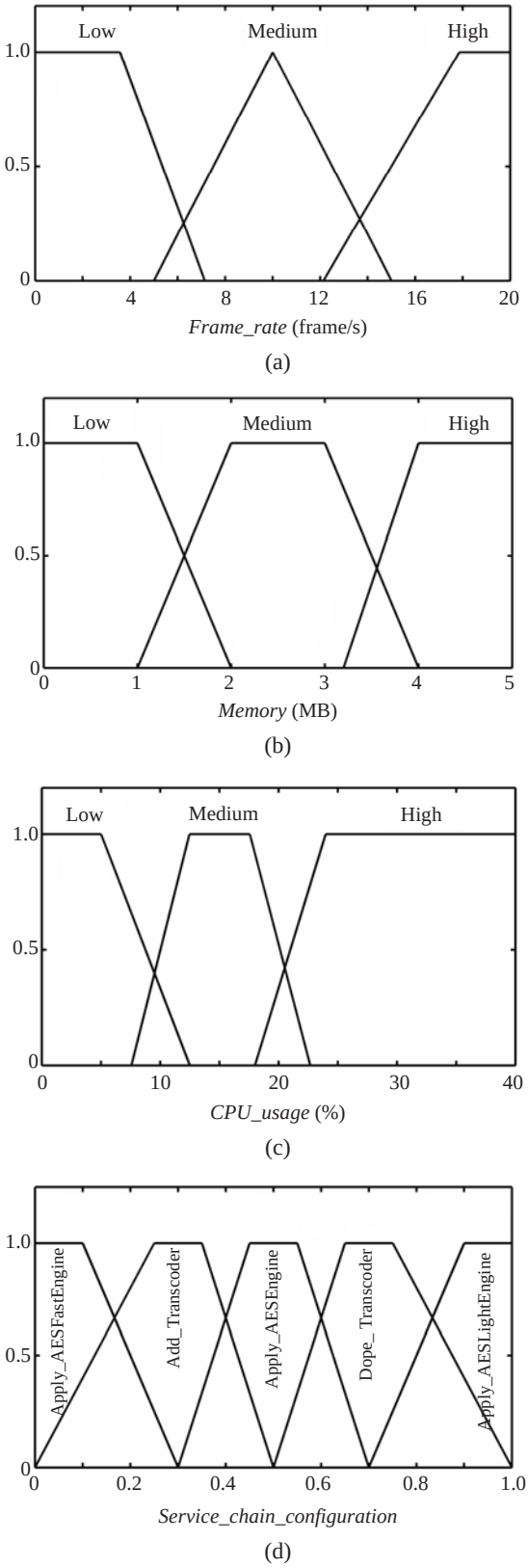


**Fig.6 In the SC-FLC, membership functions for (a) *Frame_rate*; (b) *Memory*; (c) *CPU_usage*; (d) *Service_chain_configuration***

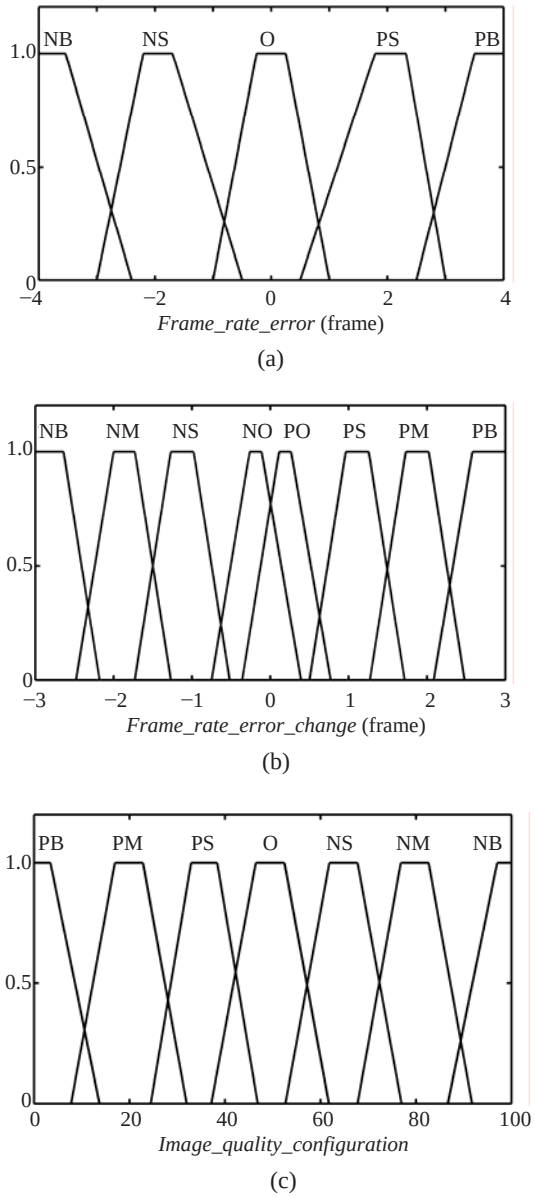


**Fig.7 In the QoSP-FLC, membership functions for (a) *Frame_rate_error*; (b) *Frame_rate_error_change*; (c) *Image_quality_configuration***

## Implementation and setup for experiments

We carried out the simulation experiments on the wireless video monitor application to validate our approach. We have developed a prototype of the wireless video monitor application and implemented service configuration approaches by using Eclipse 3.2.0 with the support of EclipseME 1.6.5 and Sony Ericsson SDK 2.2.4 for J2ME (Sony Ericsson, 2005). We performed the experiments with our approach along with the other two typical configuration

approaches, namely: Dynamic Service Chain Reconfiguration (DSCR), which has been applied to implement QoS adaptation in distributed or mobile applications (Li and Nahrstedt, 1999; Chan and Chuang, 2003); Dynamic QoS Parameter Adjustment (DQPA), which has been employed to support the mobile QoS management framework (Chuang and Chan, 2006). As mentioned above, our approach is a Comprehensive Service Configuration (CSC) which integrates these two approaches in the configuration process. To facilitate comparison, we implement all three approaches using the fuzzy logic control method with the assistance of an open source fuzzy inference engine for Java (Sazonov, 2000). Thus, the service configurator of DSCR simply includes an SC-FLC, while the service configurator of DSCR includes a QoSP-FLC. The input/output variables, membership functions and fuzzy rules in the three configuration approaches are the same as those presented in the previous section.

Our testbed is built on three machines: a Sony Ericsson Z800 emulator on an IBM R40e notebook is used as a mobile-end, and two DELL 4600 PCs work as a server and a router, respectively. The router running NIST net (NIST, 2005) can simulate various variations of network bandwidth, and the mobile-end is connected with the server by the router to emulate a wireless environment. In the experiments, suppose the user-specified target value of frame rate $q_0$ equals 10 frame/s. In CSC, the floors and ceilings of the error of frame rate in the configuration algorithm are set as: $\lambda_0$=−1.5 frames, $\lambda_1$=1.5 frames, $\lambda_2$=−2.5 frames and $\lambda_3$=2.5 frames. The application's service chain in DQPA is fixed and contains an image capture component, an image transcoder, an AESFastEngine and a network transmitter, while the image level of image capture in DSCR is fixed at 50. To facilitate repeated tests, we use 2 kB, 160×120 JPEG images with medium complexity as original images. The simulation time is 160 s. A configuration instruction is produced per 60 ms by the service configurator, and the sampling frequency of the frame rate is 50 Hz. According to Eq.(4), the average absolute deviation of the frame rate is considered as a performance metric for service configuration approaches. As mentioned above, the frame rate is the critical QoS of the application. Hence, the average frame rate is treated as the other performance metric.

In the wireless video monitor application, the variations of bandwidth have great impact on the frame rate. To test our approach fully, we need to conduct experiments in environments with different bandwidths. For this purpose, we design two scenarios: scenario I: a user walks around an area with stable wireless network signals; scenario II: a user enters and then leaves an area with unstable wireless network signals. The variations of bandwidth in the two scenarios are shown in Fig.8.

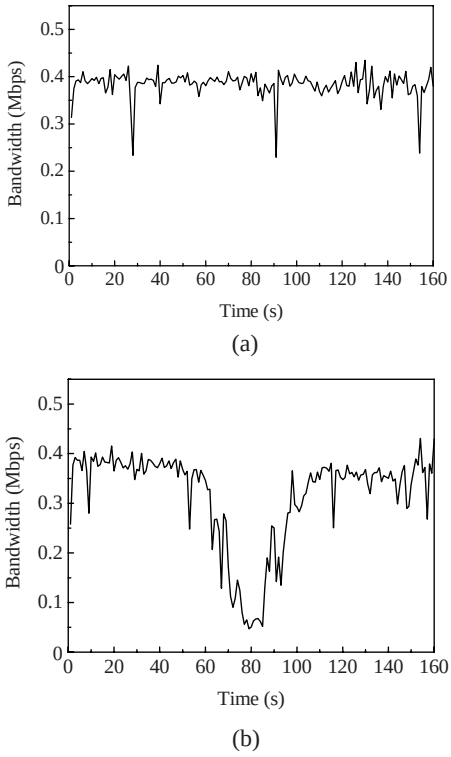


**Fig.8 Bandwidth variations in (a) scenario I; (b) scenario II**

## Results analysis

Figs.9a and 9b illustrate the results of the experiments conducted in scenarios I and II, respectively. Using our approach, the average frame rate in scenario I reaches 9.64 frame/s, up from 8.20 frame/s with DSCR and 8.77 frame/s with DQPA (Fig.9a). Moreover, the average absolute deviation of the frame rate reaches 1.67 frames, which is less than the 2.20 frames with DSCR and the 1.99 frames with DSCR. Fig.9b shows the similar characteristics in scenario II. Comparing Fig.9a with Fig.9b, we find that, the serious bandwidth fluctuation of scenario II causes a slight decline in the performance of our approach and

a sharp decline in the performance of DQPA. With our approach, the average frame rate decreases by 1.7% while with DQPA, it decreases by 10%. The average absolute deviation increases by 12.6% with our approach and by 24.6% with DQPA. In scenario I, DQPA performs better than DSCR. In scenario II, the situation is reversed. This is mainly because the serious bandwidth fluctuation of scenario II causes great changes in the frame rate. As stated previously, DSCR with service chain reconfiguration is suitable for dealing with the major changes of QoS in scenario II, while DQPA with QoS parameter adjustment is suitable for handling the minor changes of QoS in scenario I. Our approach can flexibly apply the two configuration strategies in the configuration process. Hence, our approach outperforms DSCR and DQPA in both scenarios. Also in scenario II, note that, when the bandwidth drops to a low level (below 0.256 Mbps), the frame rate with our approach is below 7.5 frame/s for only 12 s (from the 67th s to the 79th s), which is much shorter than the 24 s (from the 69th s to the 93rd s) with DSCR and the 31 s (from the 66th s to the 97th s) with DQPA.

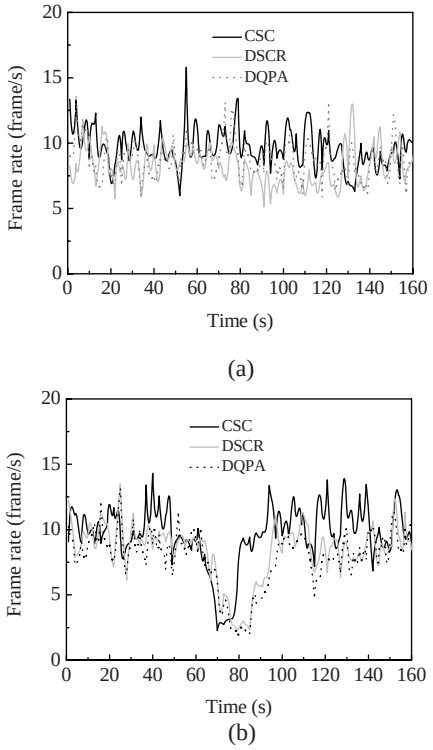


**Fig.9 Frame rate in scenario (a) I and (b) II**

We conducted other experiments to study the application's resource utilization at the mobile-end. Due to space limitation, we discuss the results only of experiments in scenario II. The resource cost at the mobile-end mainly includes battery consumption, memory occupation and CPU utilization in image transcoding, encryption and network communication. Though it is difficult to directly measure the resource cost in a simulated mobile-end, we can investigate specific indicators that reflect the resource cost. For example, throughput is almost linear to the resource cost in network communication. As for the image transcoder and encryption engines, execution time is the metric for their resource cost. Tables 5 and 6 record the results of the experiments. From Table 5, we find that, with our approach, the throughput that mainly includes sent context message, received configuration instructions and transmitted image data is less than that with DSCR and DQPA. We also note two results: (1) with our approach, the data size of the received configuration instructions is larger than that of the other two approaches; (2) with DQPA, the data size of context message is smaller than that with CSC and DSCR. The first result is because with our approach, the mobile-end receives two configuration instructions from the service configuration in one configuration iteration when the error of frame rate is out of the range $[\lambda_2, \lambda_3]$ for the SC-FLC in the configuration algorithm. But in any case of DSCR and DQPA, the mobile-end receives only one configuration instruction in one configuration iteration. The second result comes from the fact that CSC and DSCR take the context associated with frame rate, CPU usage and memory rather than simple frame rate as in DQPA. Table 6 shows that, with our approach, the execution times of the image transcoder and AESFastEngine are shorter than those with DSCR and DQPA, while the execution times of the AESEngine and AESLightEngine are longer than those with DSCR and DQPA. As mentioned above, among the encryption engines, the AESFastEngine consumes the most resource, the AESLightEngine consumes the least and the AESEngine is intermediate. With each approach, the total execution time of the three encryption engines is equal to 160 s; therefore, we know that the resource cost of encryption engines with our approach is the least of the three approaches.

**Table 5 Throughput in network communication**

| Configuration approach | Throughput (kB) | | | |
|---|---|---|---|---|
| | Context message | Configuration instruction | Image data | Total |
| DQPA | 13.34 | 10.67 | 2497.43 | 2512.44 |
| DSCR | 40 | 10.67 | 2572.62 | 2623.29 |
| CSC | 40 | 13.72 | 2434.78 | 2488.50 |

DQPA: Dynamic QoS Parameter Adjustment; DSCR: Dynamic Service Chain Reconfiguration; CSC: Comprehensive Service Configuration

**Table 6 Execution time of service components**

| Configuration approach | Execution time (s) | | | | |
|---|---|---|---|---|---|
| | Image capture | Image transcoder | AES-Engine | AESFast-Engine | AESLight-Engine |
| DQPA | 160 | 160 | 0 | 160 | 0 |
| DSCR | 160 | 48.6 | 100.3 | 48.4 | 11.3 |
| CSC | 160 | 43.5 | 112.5 | 30.8 | 16.7 |

DQPA: Dynamic QoS Parameter Adjustment; DSCR: Dynamic Service Chain Reconfiguration; CSC: Comprehensive Service Configuration

## DISCUSSION

Though fuzzy logic control employed in our approach has shown its effectiveness in the service configuration process, the large number of QoS parameters and service chain reconfiguration options in UbiComp applications would cause the rule explosion problem. This leads to a rapid increase in resource cost and a sharp decline in performance. However, the problem can be solved by organizing related linguistic variables with hierarchical structure and reducing the number of rules drastically with hierarchical inference (Yager, 1998; Wang, 1999). Moreover, the settings of our approach, such as the fuzzy rules, the membership functions, as well as the floors and ceilings of QoS error are application-specific. It is difficult to achieve good performance consistently with the fixed settings in various UbiComp environments. With artificial intelligence methods, such as the Neuro-Fuzzy model (Kim and Kasabov, 1999), we believe that self-adjustment of the settings can be implemented to improve the adaptation of our approach.

## CONCLUSION

This study presents a novel service configuration approach that can provide adaptive QoS management for the applications in mobile and resource-limited UbiComp environments. We map dynamic service configuration process to a control model which aims to control QoS variations on minimal level with less resource cost. To handle different QoS variations, a service configurator is designed as a controller which includes two FLCs representing different configuration strategies. A configuration algorithm is developed to flexibly employ the two FLCs depending on the error of critical QoS. We performed simulation experiments to validate our approach. The experimental results suggested that our approach outperforms DSCR and DQPA in terms of average frame rate and average absolute deviation of frame rate. Moreover, our approach has a lower resource cost and higher resource utilization than the other two approaches. In future work, we will extend our approach to QoS management for the applications with multiple critical QoS attributes.

## References

Bouncy Castle, 2005. Bouncy Castle Crypto Package for J2ME. Available from: http://www.bouncycast-le.org/java.html [Accessed 2006-09-27].

Chan, A.T.S., Chuang, S.N., 2003. MobiPADS: A reflective middleware for context-aware mobile computing. *IEEE Trans. Software Eng.*, **29**(12):1072-1085. [doi:10.1109/TSE.2003.1265522]

Chuang, S.N., Chan, A.T.S., 2006. MobiPADS++: A Mobile QoS Middleware Based on Hierarchical Fuzzy Control. Proc. IEEE Int. Conf. on Fuzzy Systems, Vancouver, BC, Canada. IEEE, Piscataway, USA, p.2223-2230. [doi:10.119/FUZZY.2006.1682009]

Dey, A.K., Salber, D., Abowd, G.D., 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Comput. Interact. J.*, **16**(2-4):97-166. [doi:10.1207/S15327051HCI16234_02]

Forum Nokia, 2007. J2ME Developer's Library 1.4. Available from: http://www.forum.nokia.com [Accessed 2007-08-16].

Giaglis, G.M., Pateli, A., Fouskas, K., Kourouthanassis, P., Tsamakos, A., 2002. On the Potential Use of Mobile Positioning Technologies in Indoor Environments. Proc. Int. Electronic Commerce Conf., Bled, Slovernia, p.413-429.

Gu, X.H., Nahrstedt, K., 2002. Dynamic QoS-Aware Multimedia Service Configuration in Ubiquitous Computing Environments. Proc. Int. Conf. on Distributed Computing Systems, Vienna, Austria. IEEE, Piscataway, USA, p.311-318. [doi:10.1109/ICDCS.2002.1022268]

Huang, L., Kumar, S., Kuo, C.C.J., 2004. Adaptive resource allocation for multimedia QoS management in wireless

networks. *IEEE Trans. Vehic. Technol.*, **53**(2):547-558. [doi:10.1109/TVT.2003.823290]

JCP (Java Community Process), 2005. JSR 234: Advanced Multimedia Supplements. Available from: http://jcp.org/ [Accessed 2007-07-15].

Kim, J., Kasabov, N., 1999. HyFIS: adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems. *Neural Netw.*, **12**(9):1301-1319. [doi:10.1016/S0893-6080(99)00067-2]

Kourouthanassis, P., Roussos, G., 2003. Developing consumer-friendly pervasive retail systems. *IEEE Perv. Comp.*, **2**(2):32-39. [doi:10.1109/MPRV.2003.1203751]

Li, B., Nahrstedt, K., 1999. A control-based middleware framework for quality-of-service adaptations. *IEEE J. Select. Areas Commun.*, **17**(9):1632-1650. [doi:10.1109/49.790486]

Marsh, A., 2002. The E-CARE Project: Removing the Wires. Proc. Int. Conf. on Computational Science, London, UK. Springer, Berlin, Germany, p.1012-1018.

NIST (National Institution of Standards and Technology), 2005. NIST Net. Available from: http://snad.ncsl.nist.gov/itg/nistnet [Accessed 2006-05-04].

Roussos, G., Marsh, A.J., Maglavera, S., 2005. Enabling pervasive computing with smart phones. *IEEE Perv. Comp.*, **4**(2):20-27. [doi:10.1109/MPRV.2005.30]

Sazonov, E., 2000. Open Source Fuzzy Inference Engine for Java. Available from: http://people.clarkson.edu/~esazonov/FuzzyEngine [Accessed 2007-08-12].

Sony Ericsson, 2005. Sony Ericsson SDK 2.2.4 for J2ME. Available from: http://developer.sonyericsson.com [Accessed 2006-07-18].

Su, C.Y., Stepanenko, Y., 1994. Adaptive control of a class of nonlinear systems with fuzzy logic. *IEEE Trans. Fuzzy Syst.*, **2**(4):285-294. [doi:10.1109/91.324808]

Toysoft, 2005. LightNzip Version 2.6. Available from: http://www.toysoft.ca/lightnzip.html [Accessed 2006-06-09].

Tsay, D.L., Chung, H.Y., Lee, C.J., 1999. The adaptive control of nonlinear systems using the sugeno-type of fuzzy logic. *IEEE Trans. Fuzzy Syst.*, **7**(2):225-229. [doi:10.1109/91.755402]

Wang, L.X., 1999. Analysis and design of hierarchical fuzzy systems. *IEEE Trans. Fuzzy Syst.*, **7**(5):617-624. [doi:10.1109/91.797984]

World Wide Web Consortium, 2007. Web Services Description Language Version 2.0 Part 1: Core Language. Available from: http://www.w3.org./TR/2007/REC-wsdl20-20070626 [Accessed 2007-08-15].

Xu, D., Wichadakul, D., Nahrstedt, K., 2000. Multimedia Service Configuration and Reservation in Heterogeneous Environments. Proc. IEEE Int. Conf. on Distributed Computing Systems, Taipei, Taiwan. IEEE, Piscataway, USA, p.512-521. [doi:10.1109/ICDCS.2000.840964]

Yager, R.R., 1998. On the Construction of Hierarchical Fuzzy Systems Models. *IEEE Trans. Syst. Man Cybern.*, **28**(1):55-66. [doi:10.1109/5326.661090]

Zeng, L., Benatallah, B., 2004. QoS-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, **30**(5):311-327. [doi:10.1109/TSE.2004.11]