An Event-Driven, User-Centric, QoS-aware Middleware Framework for Ubiquitous Multimedia Applications

Xiaohui Gu, Klara Nahrstedt
Department of Computer Science Department
University of Illinois at Urbana-Champaign, Urbana, IL 61801
{xgu, klara} @cs.uiuc.edu

ABSTRACT

In this paper, we present a novel QoS-aware middleware framework, that seamlessly enables ubiquitous delivery of QoS-aware multimedia applications. The QoS management functions are triggered by not only resource fluctuations but also the user's behavior, movement or even light and time. Those activation events are described using asynchronous messages encoded in XML format. Moreover, the QoS-aware middleware services are no longer targeted to applications or devices but users. The QoS management functions are individualized according to the user's preferences and life routines. We have implemented a prototype of our middleware framework as part of the Gaia OS [5], an active space enabling infrastructure. Our experiments with the ubiquitous multimedia streaming application show the soundness of our framework.

1. INTRODUCTION

Future computing environment has been envisioned as ubiquitous.[10] It consists of devices that are diverse in size, capability and power consumption. It promotes the proliferation of embedded devices specialized in specific tasks (e.g., remote sensor, tracking camera, wall display). The computing systems have been extended to the whole physical space rather than limited to a single desktop computer. A single user may possess multiple devices and frequently move from one machine to another for the purpose of convenience. On the other hand, the same computing systems may be shared by different users and should be personalized to meet the users' preference. Many emerging multimedia applications, such as video-on-demand, visual tracking, video conferencing, are being deployed in such a ubiquitous computing environment. The traditional QoS-aware middleware framework, based on the client-server model and synchronous semantics, can no longer suffice. Therefore, a new challenge in multimedia middleware research is to seamlessly provide QoS for the *ubiquitous* multimedia applications.

Different middleware frameworks have been built to provide QoS for multimedia applications. In [8], an integrated quality of service architecture (QoS-A) is proposed to enable QoS specification and provisioning for multimedia applications over ATM networks. However, QoS-A mainly focuses on the management of underlying resources and does not provide sufficient application-level adaptability and configurability to accommodate the diversity of ubiquitous multimedia applications. In [1], the reflective approach is proposed to provide dynamic QoS management functions of QoS monitoring and adaptation in middleware platforms. In the QuO [9] project, a more declarative approach, based on the aspect-oriented programming, is proposed to specify different aspects of QoS support using a set of specialized languages. Although both approaches provide a flexible QoS management framework, they do not address the new challenges, such as user mobility, device heterogeneity, posed by the ubiquitous computing environment.

In this paper, we present a novel middleware framework, which enables ubiquitous delivery of QoS-aware multimedia applications. The framework is based on our previous multimedia middleware research: (1) *Agilos* [4], a control-based middleware for QoS adaptations; (2) $2K^Q$ [7], a unified reconfigurable QoS-aware middleware; and (3) *SMART* [2], a scalable middleware. This paper focuses on addressing the new challenges brought by the ubiquitous computing environment. First, we believe that QoS management functions should be practiced in a much broader context and in an active and asynchronous manner. Therefore, we replace the passive, stand alone QoS monitoring module with an *active distributed QoS event manager*. Different context information, such as the user's behavior, movement as well as resource fluctuation, is encapsulated in XML-encoded messages and sent to the *event manager*, which in turn triggers different QoS management components (e.g., adaptor, (re)configurator, scheduler). Second, QoS management functions should be personalized according to the user's preferences and life routines. Thus, we introduce the *user QoS profiler* to automatically derive and predict the user's preference and behavior based on AI methods for making decisions (e.g., Artificial Neural Networks) [6].

The rest of the paper is organized as follows. Section 2 presents the overall middleware architecture. Section 3 describes the distributed QoS event manager. Section 4 presents the user QoS profiler and its algorithms. Section 5 presents the experimental results from our prototype. Section 6 concludes this paper.

2. MIDDLEWARE ARCHITECTURE

The overall middleware architecture is illustrated in Figure **??**. Our QoS-aware middleware framework is built on top of the *Unified Object Bus* [5], which makes our QoS provisioning solution apply to any existing middleware platforms such as CORBA, DCOM and Java RMI. Similar to the approaches presented in [9, 1], our solution promotes the separation of concerns to ease the development of QoS-aware ubiquitous multimedia applications. Traditional application developers write functional code for different multimedia application components, such as video player, transcoder. QoS developers create QoS specifications, indicating different application-specific QoS requirements and policies. However, different from the previous approaches, we use XML as our QoS specification language to make our solution most applicable. Moreover, the QoS specifications are considered in a broader context, namely the ubiquitous computing environment. We have developed a visual QoS programming environment, called *QoSTalk* [3], assisting QoS developers to generate such QoS specifications easily.

The QoS-aware middleware components are (1) *QoS controllers*, (2) *QoS event manager*, (3) *QoS specification parser*, and (4) *user QoS profiler*. The *QoS controllers* provide different QoS management functions (e.g., (re)configuration, adaptation, scheduling) and may invoke the QoS-aware resource management functions (e.g., resource reservations). The QoS event manager keeps track of different context information and triggers the proper *QoS controllers* into action. The *QoS specification parser* [3] translates the application specific policies into desired data structures by the *QoS event manager*, which steers the QoS management towards the satisfaction of application-specific QoS criteria. The *user QoS profiler* automatically derives the user's preferences and life routines, based on the user's feedbacks and experience, which are used to personalize the QoS management.

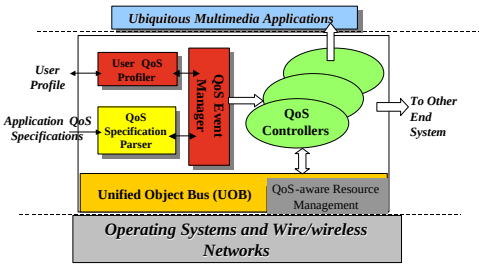

Figure 1: Middleware Architecture Overview.

3. DISTRIBUTED QOS EVENT MANAGER

The overall architecture of the distributed QoS event management is illustrated in Figure 1. Each active space [1] domain contains one *event server*, which stores local events and obtains required non-local events from other domain's event servers. The events are text-based messages formatted as XML. Each event server maintains a list of (event content, time stamp) pairs, which form an event space. The event server periodically deletes the oblivious events according to the time stamps.

[1]We use active space to refer to the ubiquitous computing environment.

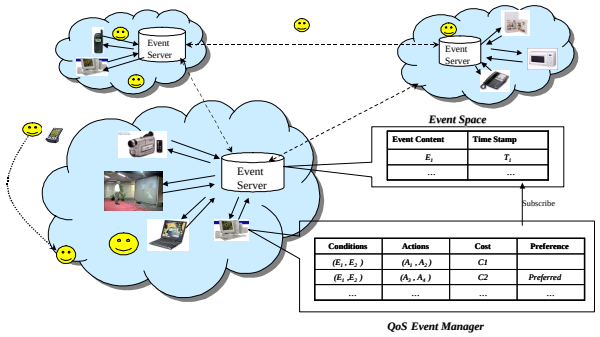

Figure 2: Distributed QoS Event Management.

The *QoS event manager* at each ubiquitous client subscribes to its local event server for the events that are of interest. It may also receive events from the entities of the local host (e.g., different resource brokers, time alarms). Each QoS event manager maintains a tuple space that is changed dynamically, during runtime, according to the current active applications and their users. Each tuple contains four parts: (1) conditions, (2) actions, (3) cost, and (4) preference. The *conditions* define the events that will activate some QoS controllers when they happen simultaneously. The *actions* specify the QoS controllers to be activated and their desired input parameters. The *cost* indicates the required system resources and energy to finish the QoS controlling operations. Finally, the *preference* is used to personalize the QoS operations. The *QoS event manager* acquires those tuples from the *QoS specification parser* and *user QoS profiler*, for application and user specific policies, respectively.

In Figure 1, for example, if events E_1 (e.g., cpu load is low) and E_2 (e.g., network load is high) both happen, then there are two possible QoS controlling operations, A_1 (e.g., drop B/P frames) and A_2 (e.g., add compression), with cost C_1 and C_2 respectively. The QoS event manager will first choose the suitable operation according to the cost. If both operations' costs are affordable by the current systems, the user-preferred one will be chosen. Now, let us consider a typical example in the active space. Usually, multiple big wall displays are driven by a single powerful PC, which is hidden behind a door. The user may focus on different displays while he or she moves around. The QoS controller is desired to reserve a suitable amount of CPU and bandwidth for the process that sends videos to the display where the user focuses. The location sensors could detect the current location of the user and generate a "location change" event when the user moves. Then the QoS event manager triggers the related QoS controllers to reserve CPU and bandwidth for the proper process.

The advantages of the active *distributed QoS event manager* are the following: (1) it can handle much richer context information than the passive, stand alone QoS monitors; (2) it promotes the separation of concerns to relieve the QoS controllers of the burden of handling complex events; (3) it enables loose coupling between entities to ease the development and management of ubiquitous multimedia applications; and (4) it allows the QoS management functions to be changed and personalized dynamically during runtime.

4. USER QOS PROFILER

Day	Time	Location	Device	Multimedia Request
Every Day	8:00PM	Home	Wall Display 1	Watch CNN Sports News
Every Day	5:00PM	On the Train	Handheld PC 1	Video Phone with Wife
Monday	10:00AM	Conference Room	Wall Display 2	Distance Learning

Figure 3: Examples of User Request Profile.

CPU Load	Network Load	Battery Life	Movement	Light	Actions	Satisfaction
Low	High	__	__	Bright	Drop Frames	30%
Low	High	__	__	Bright	Add Compression	90%
Low	Low	20%	__	__	Slow down Processor Speed	90%

Figure 4: Examples of User Feedbacks to QoS Services.

The *user QoS profiler* achieves the personalization of QoS-aware multimedia delivery by maintaining a user profile, which can move between active spaces. The user profile includes two parts: (1) user request model; and (2) user preference model. Figure 2 gives some examples of user requests. The first line of the table specifies that the user usually watches the CNN sports news at 8:00pm on wall-display1 at home. Those information are very helpful for using the *predictive* QoS management functions, such as *advance resource reservation*, *prefetching* and *advance service discovery*. The second part of the user profile reflects the user's preference and is derived based on user's feedbacks. Whenever some QoS controllers are activated in response to certain events, the *user QoS profiler* sends a notification message to the user and requests a feedback. The user may give the feedback in the form of a percentage, which represents the degree of his or her satisfaction about the QoS management services. Figure 3 shows some examples of user feedbacks. For instance, the first two lines indicate that the user prefers "Add Compression" to "Drop Frames" under the condition of light CPU load and heavy network traffic. The last line tells the *user QoS profiler* that the user prefers "slow down processor speed" in case of the low battery of the mobile device.

The contents of the user profile can be very complex and vary dynamically during runtime. Thus, instead of saving all the instances of the user's behaviors and preferences, the *user QoS profiler* maintains two prediction functions, f_b and f_p, for each user. The input parameters of the function f_b specifies the environment information like "time", "date", "location", "weather", etc. The output parameters specify the possibilities of invoking certain multimedia applications on some devices. For instance, the first element in the output vector may represent the possibility of requesting the *video-on-demand* application on the *wall display1* under certain circumstances. The input of the other prediction function f_p describes the context events and QoS management functions to be activated. The output of f_p is a percentage indicating the degree of the user's satisfaction. The

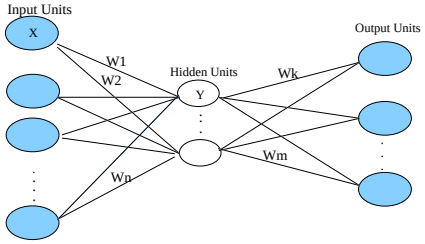

Figure 5: Three Layer Neural Networks.

Learning the user behavior prediction function f_b

- Create a feed-forward network with n_{in} input units, n_{hidden} hidden units, and n_{out} output units with learning rate η (e.g., 0.05)
- Initialize all network weights to small ramdom numbers
- **For** each user request $\boldsymbol{b}$ = (0,…,1,…0) and environment point $\boldsymbol{e}$ **Do**

//Calculate the predicted behavior using f_b

$b' = f_b(e)$;

// Update the neural network backward through the network

For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow b'_k(1-b'_k)(b_k-b'_k);$$

For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow b'_h(1-b'_h)\sum_{k \in outputs} w_{kh}\delta_k$$

Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where $\Delta w_{ji} = \eta\, \delta_j b_{ji}$

Figure 6: Learning Algorithm for the Behavior Prediction Function.

above concepts are formalized as following:

$$Environment\ \ Space: E = e_1 \times e_2 \times ... \times e_n \quad (1)$$

$$Behavior\ \ Space: B = b_1 \times b_2 \times ... \times b_m \quad (2)$$

$$Behavior\ \ Prediction: f_b : E \rightarrow B \quad (3)$$

$$Condition\ Action\ Space: CA = ca_1 \times ca_2 \times ... \times ca_k \quad (4)$$

$$Preference\ \ Prediction: f_p : CA \rightarrow [0,1] \quad (5)$$

The *user QoS profiler* automatically learns the above two prediction functions, for each user, from his or her experience and feedbacks. Currently, we use the Artificial Neural Network (ANN) [6] learning methods in the *user QoS profiler* because ANN supports online learning and fast evaluation. We model the above two functions as the three layer artificial neural networks, illustrated in Figure 4. Each node represents a computation function (e.g., step function, sigmoid function). The link between two nodes, X and Y, means that the output value of X will be used as input of Y. The weight W_1 on the link determines the contribution of the input value from X to the output of Y. Once the structure of the neural network is determined, its output will be decided only by those weights. Thus, the responsibility of the *user QoS profiler* is to learn the set of weights, which can best predict the user's behavior and preferences. Figure 5 shows the algorithm of the *user QoS profiler* for learning the behavior prediction function f_b. The learning algorithm for the preference prediction function is similar to that.

5. EXPERIMENTAL RESULTS

We have implemented a prototype of our QoS-aware middleware framework. The QoS-aware middleware components, namely different *QoS controllers*, *QoS event manager*, *User QoS Profiler*, *QoS Specification Parser*, are all implemented as *Unified Components* [5] (CORBA objects in the current implementation). The current prototype runs on both Solaris and Windows 2000. A simplified version can also run on Windows CE. The multimedia service components are also implemented as CORBA objects. We have implemented the distributed *video on demand* (VoD) application on top of our middleware framework.

We evaluate the effectiveness of the active *distributed QoS event manager* by using the VoD service in a ubiquitous manner. Figure 6 shows the control actions generated by the *QoS event manager* at their respective starting times. The experiment results show that our framework can seamlessly provide the ubiquitous VoD service with minimum user level QoS violations.

Start Time (sec)	*Events*	*Control Actions from the QoS Event Manager*
10	Start *VoD* on the Laptop connected with 100Mbps Ethernet	Trigger Configurator: "*MPEG Video Server → MPEG Video Player*",
320	High CPU load	Trigger Reconfigurator: "*Insert the discriminating Frame Dropper (B/P frames only)*"
539	Switch to 10Mbps Wireless LAN	Trigger Adaptor: "*MPEG Video Server :: DecreaseFrameRate* "
610	Migrate the session to the Handheld PC	Trigger (1) Mobility Manager: "*SaveStates (e.g., frame number)*"; (2) Configurator: "*MPEG Video Server → MPEG to Bitmap Transcoder Gateway → Bitmap Player*"; (3) Mobility Manager: "*RecoverStates (e.g., frame number)*"

Figure 7: Control Actions generated by the QoS Event Manager.

Storage Size						Processing Time / Prediction	Prediction Accuracy
Executable	f_B	f_P	User Behavior Profile	User Preference Profile	Storage Saving	0.1 ms	85%
1.28 MB	3.3 KB	2.1 KB	3.3 MB	0.8 MB	2.8 MB		

Figure 8: Experiment Results of the User QoS Profiler Implementation.

The *user QoS profiler* is implemented in C++, based on the back-propagation algorithm [6] illustrated in Figure 5. Some experiment results are shown in Figure 7. The size of the executable is 1.28MB. The total storage for the two prediction functions is 5.4KB. We test the *user QoS profiler* on a naive human subject and record 30000 instances (examples) of the user's behaviors and 5000 instances of the user's preferences in two files. The sizes of the two files are 3.3MB and 0.8MB respectively. The average processing time of the two prediction functions is $0.1ms$ and the prediction accuracy is 85%.

6. CONCLUSION AND FUTURE WORK

Emerging ubiquitous multimedia applications bring new challenges to the multimedia middleware research. In this paper, we present a novel QoS-aware middleware architecture to address the challenges. The major contributions of the framework are: (1) replacing the previous passive QoS monitor with an *active distributed QoS event manager*, so that the QoS management functions can be seamlessly integrated into the ubiquitous multimedia applications; and (2) introducing the *user QoS profiler* to automatically customize, and thus improve the QoS management for a particular user. In the future, we will build and test more ubiquitous multimedia applications on top of our QoS-aware middleware framework to verify and improve our systems.

7. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under contract number 9870736, NSF CISE Infrastructure grant under contract number NSF EIA 99-72884, and NASA grant under contract number NASA NAG 2-1250.

8. REFERENCES

[1] G. S. Blair, A. Andersen, L. Blair, and G. Coulson. The Role of Reflection in Supporting Dynamic QoS Management Functions. *Proceedings of the 7th IEEE International Workshop on Quality of Service (IWQoS'99)*, June 1999.

[2] Y. Cui, D. Xu, and K. Nahrstedt. SMART: A Scalable Middleware Solution for Ubiquitous Multimedia Service Delivery. *Proceedings of IEEE International Conference on Multimedia and Expo 2001*, Aug. 2001.

[3] X. Gu, D. Wichadakul, and K. Nahrstedt. Visual QoS Programming Environment for Ubiquitous Multimedia Services. *Proceedings of IEEE International Conference on Multimedia and Expo 2001*, Aug. 2001.

[4] B. Li and K. Nahrstedt. A control-based middleware framework for quality of service adaptation. *IEEE Journal on Selected Areas in Communication*, Sept. 1999.

[5] R. Manuel and R. H. Campbell. Gaia: Enabling Active Spaces. *9th ACM SIGOPS European Workshop*, Sept. 2000.

[6] T. M. Michell. Machine Learning. *McGraw-Hill*, 1997.

[7] K. Nahrstedt, D. Wichadakul, and D. Xu. Distributed QoS Compilation and Runtime Instantiation. *Proceedings of IEEE/IFIP International Workshop on QoS 2000 (IWQoS2000),*, June 2000.

[8] A. T.Campbell. A Quality of Service Architecture. *PhD Thesis, Computing Department, Lancaster University*, Jan. 1996.

[9] R. Vanegas, J. Zinky, J. Loyall, D. Karr, R. Schantz, and D. Bakken. QuO's Runtime Support for Quality of Service in Distributed Objects. *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, Sept. 1998.

[10] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communication of the ACM, 36(7), pp. 74-84*, 1993.