The Hong Kong Polytechnic University

Department of Computing

A MOBILE MIDDLEWARE FOR QUALITY OF SERVICE ADAPTATIONS

CHUANG SIU NAM

A thesis submitted in partial fulfillment of the requirements for

the degree of Doctor of Philosophy

Initial Submission: August 2008

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no materials previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signed)

Chuang Siu Nam (Name of student)

Abstract

Computation and networking resources in mobile operating environments are much scarcer and more dynamic than in desktop operating environments. Mobile applications can use adaptive computing to optimize the quality-of-service (QoS) delivery based on dynamic contextual situations. Fuzzy control models have been successfully applied to various distributed network QoS management systems. However, existing models are either application-specific or limited to abstract modeling and simple conceptual scenarios that do not take into account the overall scalability of these models. Specifically, the large number of QoS parameters in a mobile operating environment causes the rule-explosion problem, in which an exponential increase in the number of rules correspondingly increases the demand for processing power to infer the rules. Hierarchical fuzzy systems were introduced to reduce the number of rules using hierarchical fuzzy control, in which correlated linguistic variables are hierarchically inferred and grouped into abstract linguistic variables. In this thesis, we propose a mobile QoS management framework that uses a hierarchical fuzzy control model to support a highly extensible and structured adaptation paradigm. The proposed framework integrates several levels of QoS abstractions derived from user-perceived requirements. It also maps these abstractions to appropriate QoS resources that drive the development of mobile services that mitigate the effects of varying mobile environments. This framework provides an optimal overall service by synergistically balancing the QoS requirements of users and applications with the dynamic allocation of resources and chaining of services. Our proposal is novel in that it looks at QoS from a holistic, systematic, and pragmatic perspective. This thesis demonstrates the flexibility and efficiency of our QoS management framework in adapting to different users, applications, and platforms operating in wireless environments that are characterized by dynamic and constrained resources.

Acknowledgements

There are a lot of people I would like to thank, for they influenced me and supported me in completing this thesis, a difficult yet rewarding process.

First of all, I would like to thank my supervisor, Dr. Alvin Chan, for taking me as a graduate student. It was really good fortune for me to become his student. He has been an ideal supervisor in every aspect, both in terms of technical advice on my research and in terms of professional advice. I hope I can live up with his high standards in my future career.

I would also like to thank Martin Kyle, for his help in the preparation of this thesis. Thanks are also due to Ms. Miu Tai and Ms. Rosa Kwan, for their friendliness and approachability in solving the administrative problems on my study. I appreciate the supports provided by the departmental tech team, for their prompt responds to my frequent request for equipments, software and other resources.

My special thanks are due to my family for their continuous love, encouragement and patience. Most of all, I would like to thank my lovely fiancée, for her patience and sacrifice throughout my study. Without her support, this dissertation would never have been completed.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless mobile computing operates in a paradigm where resources (dimensions of devices, power, CPU speed and memory) are not only constrained but are often not reliably available or vary dynamically and as a result .connection may have high service costs and error rates and suffer temporary disconnections. System designers have usually responded to such resource-constraint or quality of service (QoS) challenges by simply compromising performance, usually by operating under the threshold of some presumed level of resources. But such protocols and services, operating on a resources-stable, hard-guaranteed reservation scheme, i.e., “best effort” delivery networks based on a TCP/IP protocol, are not suitable for mobile environments as they do not allow any adaptation, either in the case of deterioration or improvement in resource availability or stability. Some TCP/IP-based QoS mechanisms, such as diffServ [DiffServ08] and IntServ [Evans07], do support differentiating and integrating different types of network streams, but these mechanisms still require relatively stable fixed network environments.

These traditional reservation-based QoS management mechanisms cannot effectively deal with the scarcity and dynamic variation of resources, such as abrupt changes, that is characteristic of mobile environments. The focus of mobile QoS management has thus shifted to providing adaptive and optimized mobile services which provide augmented services and optimize protocols to match the available resources and operating context of the network. For example, a mobile Web

browser retrieving a graphic-rich Web site can ask the middleware service to progressively degrade the picture quality if it detects a drastic drop in the bandwidth availability. In this thesis, we identify and address the challenges in mobile QoS management and propose a QoS management framework that adopts a hierarchical fuzzy control model [Raju91] [Ronald98] to support a highly extensible and structured adaptation paradigm.

1.1 Motivations

To optimize service delivery under constrained environment, a primary objective of mobile QoS management is to facilitate appropriate resources allocation and, if necessary, make tradeoffs between QoS parameters. The ultimate goal is to maximize the user's perceived quality of service. The following describes the major difficulties in maximizing this perceived QoS, that is, in mobile QoS management.

1.1.1 Wireless Network Connectivity

Mobile devices typically connect to the internet via wireless links, which do not offer either the capacity or stability of wired connectivity, notwithstanding the higher bandwidth and lower error rates now being provided by wireless technologies like Bluetooth 2.0 [Bluetooth08], WiMax [WiMax08] and IEEE 802.20 [MBWA08] The lower bandwidth of wireless connectivity puts limits on the quality of multimedia application, such as video streaming and video conferencing as the QoS manager is required to negotiate with the application on a minimum maintainable bandwidth while supporting levels of prioritization and error detection/correction for different packet types within the same data stream and

at the same time dealing with arbitrary wireless connectivity degradation. Mobility can make wireless connectivity hard to maintain and this must be taken into account in mobile QoS management. Base station handoff, for example, when a mobile device accessing a wireless network moves from one base station to another, can result in the suspension of network availability. There has been a lot of work on reducing this disruption [Pahlavan00] [Campbell02], mostly focusing on the network and data link layers. Bandwidth variation, error rate variation, and even temporary disconnection can be addressed by applying QoS adaptation and by involving the application in the adaptation strategy. This would mean that rapid but mild changes in the network resources would be handled by an automated QoS adaptation mechanism but drastic changes would be handled by the mobile application, which would choose a strategy based on its execution and usage context. Similarly, improvements in network performance must also be taken into consideration so that a QoS management system must be able to re-negotiate with an application to use the newly available bandwidth, For example, when a better bandwidth situation is detected, a video streaming application can change its codec from high compression ratio- CPU intensive to moderate compression ratio less CPU intensive, freeing up CPU cycles and lengthening the battery life. This kind of capability is not present in reservation-based QoS systems

1.1.2 System Limitation

Mobile devices are made to be portable; however, portability comes at the cost of compromised functionality [Imielinski94] [Katz94] [Davies96] and power. PDA-sized handheld computers can now run Windows XP but their processors are limited by the limits of battery power., as are the display size and resolution.

Batteries are also stressed by wireless connectivity itself, as using typical WiFi connectivity can halve a PDA's power up time. A more efficient mobile QoS management system should not only efficiently allocate and utilize battery resources but enable the mobile device to shift part of its processing and networking duties to the wired side. The QoS system might also provide media transcoding services that convert the media stream into different formats, such that the network and CPU processing requirements for the mobile device could be minimized, and optimal presentation quality can be achieved by the mobile application. Interestingly, transcoding services usually degrades the media content in some aspects, e.g. resolution, frame rate and clarity, however it does not necessarily degrade the presentation quality of the media, sometime it could even improves the user-perceived presentation quality. E.g. streaming a mpeg2 video clip to a PDA is unlikely to have a satisfactory playback quality, as the bandwidth, jitter and insufficient decoding speed of PDA is unable to catch up with the requirement of the streamed media; however, before streaming to the mobile device, if there is a transcoding service that converts the stream based on the available bandwidth, processing power and display capacity, a significantly improved presentation quality can be realized.

1.1.3 Context Awareness

Context-awareness is another important issue for mobile operating environment. The mobility of the mobile device causes various coarse grain changes to the system, from lower level resource changes to high level user requirement changes. E.g. a battery operated mobile device is docked onto its docking device, such that battery life is no longer a limitation. Another example could be that the screen

brightness of a mobile device is tuned down when the user moves from an outdoor sunny area to an indoor, softly lighted area. Coarse gain changes usually have significant effect on mobile QoS, it is thus essential for the QoS management system to be able to capture these changes, and be able to dynamically readjust it QoS adaptation scheme accordingly, and communicate the changes in context to mobile applications.

1.1.4 Deducing Cost Functions and Formulating an Optimization Model

Any QoS model that seeks to effectively manage service and protocol adaptations must integrate many QoS parameters from different resources, contexts and operating systems. This makes it difficult to deduce cost functions that would allow meaningful comparisons and, consequently, formulate analytical optimization model for mobile QoS management. Mobile devices operating across diverse environments are subject to significant variation of three different types. First, quantization levels and quantitative scales may vary significantly across different system resources. Even within the same resource type, different benchmark and scale factors may be used to quantify a resource. This can lead to difficulties in producing appropriate scalings and mappings that would enable a consistent interpretation of quality of resources. Second, infrastructure support will also significantly vary across different execution contexts. Different connectivity technologies may be employed across different locations, the wireless coverage area and density of nodes may vary, the computation and networking limitations imposed by different power saving modes, and so on. Third, different mobile devices will differ in their capabilities, in terms of processor speeds, memory, storage, and operating systems, and in other ways.

1.1.5 Varying User and Application QoS Requirements

Under limited and varying resources conditions, different user can have very different preferences towards the same application, e.g. a user listening streamed audio would prefer better clarity, while another user would prefer shorter delay. Likewise, a user's preferences toward different encoding profiles of media of the same application could be very different too, e.g. a user prefers high frame-rate in watching sport video, but prefers high fidelity while watching travel video. Moreover, even for the same application and the same media, user's satisfaction level on a specific QoS parameter is not proportional, e.g. to the perception of many users, the effect of a frame-rate increase from 15 frame per second (fps) to 20 fps is more significant than increasing frame-rate from 25 fps to 30 fps. This makes it difficult to predefine static adaptation rules or policies that can continuously optimize user-perceived QoS and implies an adaptation selection process that is itself adaptive to changing user and application requirements and mandates personalized QoS profiles.

1.1.6 Balancing the Use of Resources in a Responsive Adaptation Model

While responsive adaptation is highly desirable, the limited computation resources available in mobile environments and devices make it necessary to make the process of interpretation and adaptation resource-efficient. Pervasive computing operating across diverse network domains and contexts requires a mobile QoS model that is able to uniformly represent the diverse characteristics of different QoS parameters while being scalable in supporting complex and evolving QoS requirements.

1.2 Research Challenges

The current need of mobile and pervasive computing is for a flexible and adaptive mobile QoS management model that is able to uniformly represent the diverse characteristics of different QoS parameters while being scalable so as to support complex and evolving QoS requirements. In the following, we organized the relevant real-world design problems in terms of four distinct issues: *abstraction versus generality*, *QoS mapping*, *policy configuration*, and *application involvement*.

Abstraction versus generality. A high level of abstract modeling across all QoS parameters is commonly applied in designing a generic QoS model that can accommodate issues of variation, flexibility, scalability and adaptability. However, the mathematical complexity of such models can lead to loss of generality when they are applied to actual QoS parameters. On the other hand, reducing this complexity by placing constraints on the number of QoS parameters to be modeled presents scalability problems when increased number of contexts is considered.

QoS mapping. Mapping the current resource QoS parameters and user-perceived QoS parameters to an adaptation, has been proposed as a way to obtain the most favorable QoS profile as specified in the user and application QoS requirements. However, QoS mapping is not a trivial task as it involves the prediction of the effects of adaptation options on the resource QoS parameters and the user-perceived QoS parameters under the current execution context. Moreover, many adaptation options are both discrete and have coarse ranges of application. Take for example the situation in which an audio streaming application

experiences a 30% drop in bandwidth. It would be ideal to have an adaptation option that could cope with this by downsampling the audio stream from 16-bit to 11.2-bit. But such an ideal option would not normally be available so the most appropriate adaptation is to downsample to 8-bit. This, however, would lower the bandwidth requirement by 50% and result in underusing the available bandwidth. Thus, an over-reactive adaptation mechanism can, instead of improving QoS, produce oscillations that may further degrade QoS.

Policy configuration. The configuration of QoS policies, the control rules and parameters that govern the adaptation decision process, are specified in usage scenarios where one set of policies is associated with a specific mobile application. The QoS management system uses this set of policies to enforce adaptation based on the prescribed environment dynamics for the mobile application. However, in real world situations and under various differing application scenarios, a mobile application may enlist differing QoS profiles. For example, a video streaming application can have alternative profiles for video using a range of other encoding schemes and bit rates. Moreover, it is not uncommon for a user to change his/her preferences and priorities regarding various aspects of QoS performance. Ideally, a QoS policing framework should be dynamically reconfigurable to match changes in user and application preferences during runtime.

Application involvement. Adaptation can take place either at the middleware layer or within the application. When it takes place at the middleware layer, the application is relieved of the need to monitor the environment or make adaptation decisions. This is an application-transparent approach. The middleware provides

best effort adaptation to general mobile computing and context information is completely hidden from the application. The drawback of this approach is that it significantly limits the amount of adaptation space available to the QoS middleware to optimize processing given that in the event of adverse conditions, the application itself is in the best position to make critical decisions on operating conditions and, hence, on the adaptation strategy.

1.3 Contributions

This thesis presents MobiPADS, a mobile QoS management framework that adopts a hierarchical fuzzy control model [Raju91] [Ronald98] to support a highly extensible and structured adaptation paradigm. Architecturally, the MobiPADS system uses edge proxies that are strategically placed along communication paths that may suffer from significant contextual changes and resource fluctuations. This often involves deploying MobiPADS across wireless links to facilitate adaptation processes that counteract the detrimental effects of contextual changes and resource fluctuations. The proxy structure of MobiPADS promotes rapid deployment over existing Internet architecture without the need to engage changes to routing protocols or operating systems. In order to achieve optimal user-perceived QoS, MobiPADS supports composition and reconfiguration of mobile services to mitigate the effects of adverse mobile operating environments and to support adaptation to dynamic QoS requirements. The major contributions of MobiPADS are as follows:

Allows Service composition and reconfiguration. Adaptive computing is fundamental to MobiPADS approach to achieving agile and optimized QoS

delivery in the hostile operating environments. This requires service composition and reconfiguration that can match the system dynamics and optimize the QoS perceived by user. To mitigate the effects of adverse conditions in a wireless environment, MobiPADS configures services, called mobilets, as chained service objects to provide augmented services and protocols to the underlying mobile applications. Each mobilet provides a specific functionality so a mixed combination of mobilets is often composed to provide added services that match different QoS requirements and contextual environments. By adding and removing mobilets, it is possible to dynamically reconfigure service chains during runtime to adapt to changes in QoS requirements and to the operating environment. Moreover, each mobilet can have a variety of modes of operation (profiles) to support finer adaptation levels. This mobilet service model enables flexible QoS support for rapid changing mobile environments, extending from the system resource level up to the user level.

Provides an extensible and comprehensive Hierarchical Fuzzy QoS Model. To integrate various services requirements and specifications across diverse entities requires a model that unifies the representation and characteristics of diverse QoS parameters. Because our QoS model forms a hierarchical QoS graph directly, it is able to incorporate and represent any QoS parameter by describing its relationship with other QoS parameters. Further, it also supports the addition of new QoS parameters to the existing QoS hierarchy, such that the number and variety of managed QoS parameters will not be bound by the original design. This also supports the reuse of existing QoS policies and allows updating of only the direct parent node of the new QoS factor within the hierarchy. This obviates the need to

review all the QoS policies when new input is added.

Allows meta-level QoS policy configuration. One of the challenges of regulating and enforcing quality of service provisions in mobile applications is the need to formulate robust policies that accurately captures the requirements and specifications of QoS. At the same time, they should provide a level of abstraction that is intuitive, measurable and concise enough to help and guide resource management. This is a particular advantage of the MobiPADS approach. In running a mobile application, it is not uncommon to encounter situations where the operating contexts have changed to such a degree that no reasonable adaptation can be exercised without a change in underlying QoS policy. A unique contribution of MobiPADS is the provision of meta-level adaptation mechanisms to support dynamic changes in user and application policy requirements at runtime. Both the user and application can affect the QoS policy by specifying the desired priorities and values of QoS parameters. The Membership Function Adaptation mechanism automatically adapts the fuzzy membership functions to these values while the Rule Weight Adaptation mechanism dynamically changes various rule weights to reflect the changed priorities. A very important benefit of this approach is that these two mechanisms are abstracted to a level that requires minimum input from the user or application to select favorable QoS policies. There is no need for the user or the application developer to understand and manipulate the underlying low level adaptation mechanisms.

Allows middleware-driven and application-participation adaptation. Adaptation can be applied and exercised across two extreme spectrums; one relying solely on

middleware and another where applications can drive adaptation decisions. MobiPADS supports flexible handling of adaptation decisions that promote synergized middleware-driven and application-participation adaptation. From a software engineering perspective, it is highly desirable to separate the process of adaptation to the middleware because it allows modularized handling of concerns and frees application developer from having to make adaptation decisions. However, this may result in the middleware making adaptation decisions that provide generalized configurations of services that aim to optimize QoS across *all* applications. MobiPADS differs from this in that applications are allowed some level of participation in formulating the overall adaptation policy so as to enable underlying application or even the end user to provide reflective adaptation of QoS requirements under changing contextual environments. Specifically, applications may participate in QoS parameter monitoring, QoS adaptation triggering, and QoS policing through a set of Reflective API. MobiPADS provides a reflective API for applications to inspect (through reflection) and to adapt (through reification) the behavior of the underlying system components, adaptation rules, and actions. While the dynamic adaptation of QoS middleware offers some degree of context adaptation, there are nonetheless times when mobile applications are in the best position to make critical decisions about the operating context and the associated adaptation strategy. For this reason, MobiPADS provides the mobile application with an extensive set of APIs and reflective interfaces. Through the meta-level object representation of the internal event system and service reconfiguration, a mobile application can access the contextual information, service configuration, QoS model and QoS policies of the QoS middleware, and modify these entities to obtain optimal service provision from middleware.

We have demonstrated the feasibility and efficiency of MobiPADS through extensive simulations and a prototype implementation on WinCE and shown that MobiPADS is able to adapt to the dynamics in the wireless environment by selecting the best possible mobile service combinations and maintain an optimal, balanced user perceived QoS. The results also show that the mechanisms of dynamic QoS policy configuration are effective. The middleware is able to use user/application supplied priorities and the preferred values of QoS parameters to dynamically redistribute the underlying resources to support the desired levels of perceived QoS. Our prototype implementation shows that the fuzzy based inference and adaptation engine is compact enough for the computation, memory, and battery limitations of a typical PDA device. In a typical usage setup consisting of 32 QoS parameters and 775 QoS policies (fuzzy rules), each inference takes about 0.5 second to execute with a 10-second execution interval. The overall battery life drops only by 8.4%, which is promising given the benefits of the adaptation capacity of MobiPADS.

1.4 Organization of Thesis

The rest of the thesis is organized as follows. Chapter 2 describes the four main stages in the operation of QoS management of Adaptive Mechanisms. We follow this by providing examples of a number of adaptive mechanisms that are well-suited to satisfying changing requirements during runtime. We describe these adaptive mechanisms and how they can be utilized to adapt to the dynamic QoS requirements of mobile operating environment.

Chapter 3 presents an overview to the MobiPADS middleware. In Chapter 4, we illustrated and evaluated the hierarchical fuzzy control model of MobiPADS. Chapter 5 introduces the meta-level adaptation mechanisms for supporting application participations of QoS management at various adaptation stages, including membership function adaptation, importance weight adaptation and computation reflection. Chapter 6 offers a comparative review of related work. Finally, Chapter 7 concludes this thesis and outlines some directions for future work.

Chapter 2

Background

In this chapter, we present an overview of a generic QoS management framework, its essential components and their interactions. Particularly, we discuss the impact of mobile environments on the design of various components of a generic QoS management framework. Then, we exemplify a number of adaptive mechanisms that are well-suited for fulfilling changing requirements during runtime. We look into these adaptive mechanisms to see how these mechanisms work and how they can be utilized to adapt to the dynamic QoS requirements of mobile operating environment.

2.1 QoS Management Framework

In this section, we first describe the flow of the four main QoS activities, QoS specification, QoS interpretation, admission control, and QoS control and then in the following subsections we consider this same material in greater detail, explaining the interaction sequences of these activities at four layers of the computational hierarchy, application level, middleware level, and system resource level.

Figure 2.1 shows the flow of concepts and corresponding operations. The QoS specification stage uses an abstract representation of QoS requirements to communicate to the lower level. The QoS interpretation stage compiles the high-level QoS representation and translates it to lower level QoS parameters that

map and distribute the available resources of the system based on the desired specifications and context so as to maximize the QoS availability under constrained and varying contextual environments. Admission control has three tasks: to check the QoS requirements against the available resource level; when needed, to negotiate with the application; and, if the application is admitted, to reserve resources. The final QoS activity is QoS control, in which system ensures that the QoS provision and consumption parameters follow the agreement or, if the parameters are violated, take action to adapt.

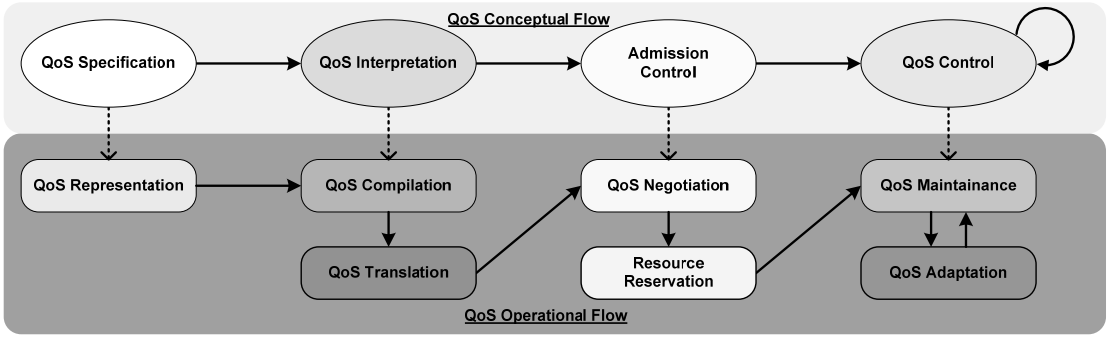

Figure 2.1 Flow of QoS Activities

2.1.1 QoS Abstractions

QoS abstractions are used to represent QoS requirements at different system levels so that there can be efficient communication between those levels. These QoS abstractions refer to requirements and parameters at every level of the computational hierarchy, from the user level down to operating system and network level. Figure 2.2 shows four layers of QoS abstraction from the user level through to the system resource level. At the user level, users can specify parameters like the responsiveness of the application, the presentation quality of the media, and the price that the user is willing to pay. At the application level, an application can

specify its bandwidth requirement, timeliness of the packet transmission, reliability and security requirements of the network channel. At the middleware level, there are QoS parameters that specify the usage of a variety of middleware services, such as filters, transcoders and different types of network channels. At the system resource level, there are QoS parameters that specify low level resource requirements, such as parameters of the TCP/IP protocol stack, CPU clock cycle, power consumption, and memory size.

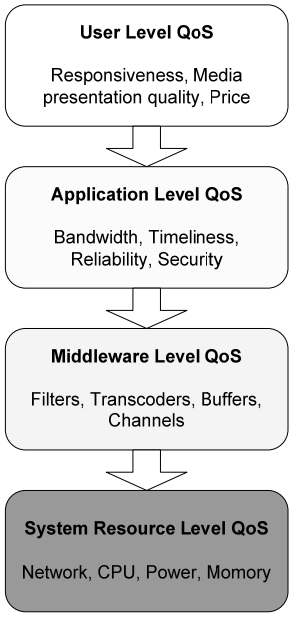

Figure 2.2 QoS Abstraction Layering from User Level to System Resource Level

2.1.2 QoS Interaction Overview

QoS management is comprised of a collection of interacting tasks that support QoS provision at four levels. Figure 2.3 shows the QoS interaction sequences in a typical QoS management framework. Note that the four QoS processes – QoS specification, QoS interpretation, admission control and QoS control --are involved in activity sequences are each of the four layers of abstraction user level, application level, middleware level, and system resource level. The gray

backgrounds indicate inter-level interactions. The white backgrounds indicate processes that are intra-level activities. The following subsections explain the interaction sequences in detail.

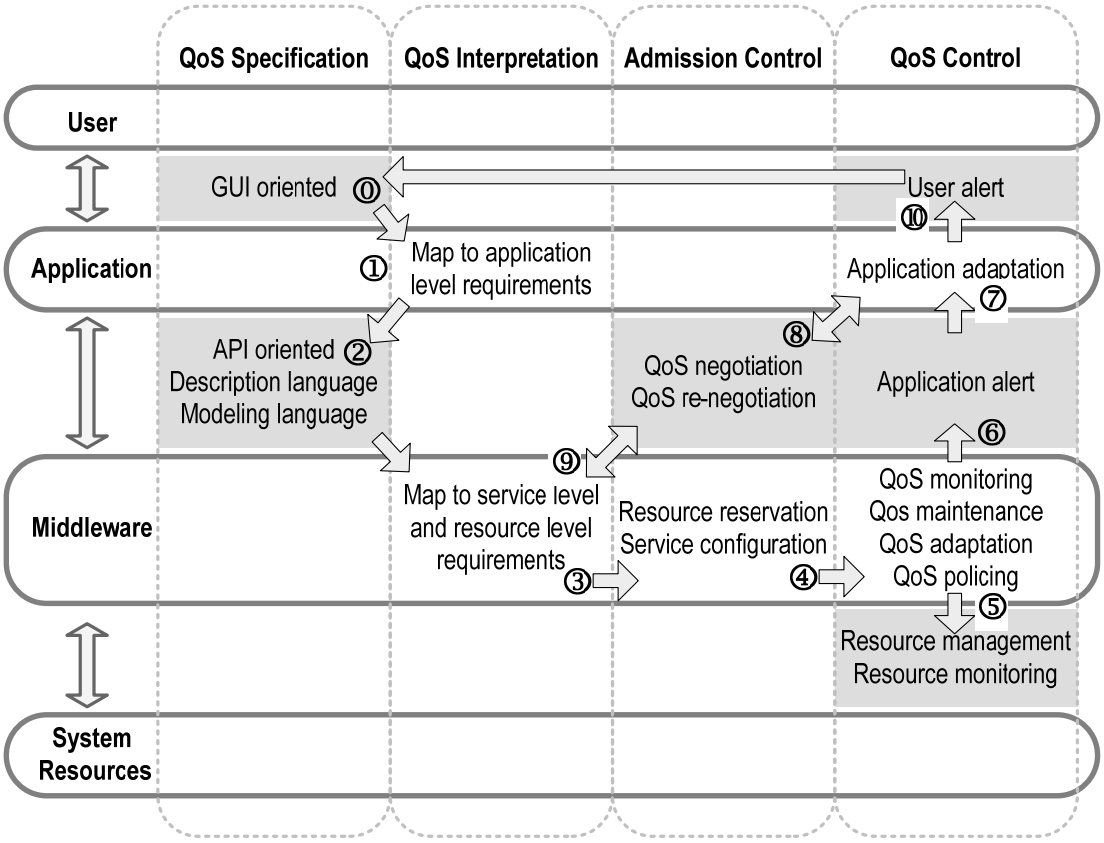

Figure 2.3 The Layered QoS Interaction Sequence

2.1.3 QoS Specification and Interpretation

The interaction sequences of QoS specification and interpretation are very closely related and so here we consider them together. Now, before an application is started, a user's preferences in relation QoS requirements are collated so as to define the user's quantitative and qualitative expectation of the service provided by the application under varying contextual environment. At the application level, the application then maps abstract user's preferences to application specific

requirements that may fulfill the required QoS. For example, a video clarity level required by the user can be translated into parameters for video resolution and color depth at the application level.

The application then communicates its mapped QoS specifications to a QoS-aware middleware, which uses the specifications to manage resources and services and satisfy the requested QoS. This process can be done in various ways, e.g. using API oriented specifications, which have the flexibility to allow the application to change its specification, or using language paradigms to decouple the specification from application development. Language paradigms can be developed locally or by extending existing standard languages, and can be either declarative or instructive. The middleware is tasked with mapping the QoS requirements to the middleware services level and resource level requirements. These translated middleware service requirements and resource requirements will at later stages form important parameters for decision making involving admission control and QoS management

2.1.4 Admission Control

Depending on the translated requirements and the currently available services and resources, the middleware can admit the application unconditionally or through a process of QoS negotiation. This process takes place between the application and the middleware, which follows the QoS specification and interpretation paradigm, and the application is admitted once they agree upon a mutually acceptable QoS specification. Upon admission, between the application and the middleware make a QoS agreement wherein the middleware has to configure corresponding services

and allocate sufficient resources. Service configuration can be in two forms, private service which pertains only to the admitted application or shared service which applies to all suitable applications. The choice between the two forms is based on the framework design, the QoS specification, and the characteristics of the individual service, e.g. whether the data are sharable or whether there are any privacy issues.

There are three types of resource allocation: 1) complete reservation, in which the system attempts to reserve a guaranteed amount of resources exclusively for the application; 2) partial reservation, also known as shared or dynamic/adaptive reservation. Partial reservation reserves only a portion of guaranteed resources, using statistical analysis and heuristics to estimate future resource consumption patterns but being mindful that it should be highly likely that of the guaranteed QoS level will be attained; 3) applying best effort delivery, which in fact means no reservation is required at all. The best effort approach does not try to reserve resources but instead applies mechanisms to control resource consumption levels so as to avoid or alleviate situations where resource availability is severely low.

2.1.5 QoS Control

Once the application is admitted and services and resources allocated, the middleware has to manage the QoS provision level and QoS consumption level so that both the application and the middleware behave so as to avoid violation of the QoS agreement during runtime. This involves applying four mechanisms: monitoring, maintenance, adaptation, and policing. The purpose of QoS maintenance is to maintain a level of resource availability and service performance

that will support the QoS agreement. This is typically done by turning service parameters, e.g. adjusting the buffer size to minimize delay and jitter in a data stream. Qos adaptation can be regarded as a more dynamic form of QoS maintenance. It takes place when there is a significant change in the operating environment that may severely affect the QoS provision level, so that the middleware has to reconfigure itself to adapt to the changes. For example, the network bandwidth may experience an abrupt decrease in availability during a Web browsing session, which significantly affects the user perceived response time. On detecting such a situation, the middleware can insert a text compression service in between the data stream to reduce the data volume of the Web pages, which reduces the response time but with the tradeoff of higher CPU usage for data compression and de-compression. The purpose of QoS policing is to ensure that and enforce that the overall consumption level is in accordance with follow the agreed QoS if the resource consumption pattern of the application is violated partially and temporarily. The available mechanisms include throttling the resource consumption, penalties on resource consumption, and notifying the violating application. QoS policing is important because it guarantees the fairness of resource allocation among the applications running on the same platform, and it also prevents poorly behaved applications affecting the execution of other well behaved applications.

Resource monitoring and management refer to the mechanisms that measure, report and control the physical resources' availability and consumption. Typical physical resources include memory, storage, bandwidth and CPU clock cycle. Resource monitoring and management is the level that tradition QoS middleware focused on,

where the attached applications have to specify the low-level QoS parameters. The recent trend in QoS middleware is to support high level application-oriented QoS specification, as it is not necessary for the majority of applications to be aware of the low-level issues. Application developers could spend more effort on what would be the desired QoS level, rather than thinking how to achieve the desired QoS level.

In the event that the operating environment no longer conforms to the limitations of the QoS agreement, the middleware is responsible for notifying the application by issuing an application alert..The middleware should not issue such alerts unless the middleware is unable to compensate for the effects of changes and their impact on the application. Whether an application alert is issued depends on the stability of the operating environment and the effectiveness of QoS management, but also on the rigidness of the QoS agreement. A rigid QoS agreement is expensive as it will trigger alerts more frequently for QoS sensitive applications as well as degrading the overall system performance, due to overheads introduced by the alert messaging and interruptions of the application for alert handling. On the other hand, alerts can also be used for middleware event notifications that the application can subscribe to. The application can thereby be more aware of the status of service configurations, resource availability and consumption, and adaptation behavior that are regulated by the middleware.

Depending on the type of application alert being sent from the middleware, the application may need to respond to do things to optimize its performance. If the alert is a subscribed event update that is relatively insignificant, e.g. battery level

drops below 50%, then the application can use this information as a reference to adjust its operation to reduce power consumption, e.g. switching off some non-critical visual effects. If the alert is indicating a violation of the QoS agreement in that it is no longer attainable by the middleware, the application will have to downgrade its QoS requirements and carry out a QoS re-negotiation with the middleware. If there is an abrupt and transient downgrading of the QoS resulting in service changes and adaptation that could push the user's experience below an acceptable service level, the application alerts and consults the user through a user alert, informing .the user of the situation and requesting a re-specification of the QoS preferences within the currently achievable ranges of QoS parameters, taking us back to the beginning of the entire process.

2.2 Adaptive Mechanisms

In this section, we provide examples of a number of adaptive mechanisms that are well-suited to satisfying changing requirements during runtime. We describe how these adaptive mechanisms work and how they can be utilized to support adaptation to the dynamic QoS requirements of mobile operating environments.

2.2.1 Aspect Oriented Programming (AOP)

Software engineers decompose software into smaller, more manageable and comprehensible parts according to certain criteria or "concerns", which might include requirements, use cases, features, data structures, and many other issues, Concerns can range from high-level abstractions like security and QoS to low-level functionalities such as caching, and failure handling. They can also be functional, such as business logics, or non-functional, such as availability and compatibility.

Some concerns, such as compatibility, usually couple with a few entities that handle the I/O of the system, yet achieve good solidity. Other concerns, such as failure handling, will interleave with many highly unrelated entities within the system.

Software developers manage software complexity by applying the principle of separation of concerns [Dijkstra76]. Programming languages support separation of concerns by using different sections for specifying the data structure and the operations that manipulate the data. Software designers also separate concerns in software design notations. UML [UML08] for example, provides different types of diagrams for separately specifying structural and functional aspects of the system. AOP separates concerns at the source code level.

Traditional object-oriented programming (OOP) captures attributes and behaviors of related entities in a class hierarchy. However, OOP faces difficulties when capturing concerns that do not fit naturally into a single class hierarchy, or even a composition of interrelated class hierarchies. In contrast, AOP supports the addressing of crosscutting issues that affect many unrelated entities, captures attributes and behaviors of these issues in a new software layer, in which an aspect module addresses a particular issue across classes of different domains within the system, thus enhances the modularity of the system beyond that of OOP.

AOP is not a replacement for OOP. Rather, it is an additional software development technique that helps solve complex problems. Unlike OOP, which has been well studied and practiced for many years, AOP is still young that lacks formal rules in identifying and isolating an aspect. As such, developers must rely

on their own judgment to model the aspect effectively and carefully. A fundamental principle in differentiating between an object-orient component and an aspect is stated in an early aspect paper [Kiczales97] by Kiczales et al.:

> *"… a property that must be implemented is:*
>
> *A component, if it can be cleanly encapsulated in a generalized procedure (i.e. object, method, procedure, API). By cleanly, we mean well localized, and easily accessed and composed as necessary. Components tend to be units of the system's functional decomposition, such as image filters, bank accounts and GUI widgets.*
>
> *An aspect, if it can not be cleanly encapsulated in a generalized procedure. Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways. Examples of aspects include memory access patterns and synchronization of concurrent objects."*

Aspects provide *crosscutting modularity* that cut across various objects of a program. By writing a single aspect module, a developer can address a specific concern that affects many parts of the program, rather than searching all over the program to find and update the related parts. In general, AOP allows developers to write code addressing crosscutting concerns once and apply it on wherever place needed within the program. References to an aspect are added at interested *join points*, which are specified by a *pointcout designator*. Specific code written to address the aspect is called an *advice*. An AOP complier can follow the references and weave the advice into the appropriate locations of the program. AOP eliminates a large amount of scattered code that addresses different concerns, so that it is much easier to maintain and upgrade a program. The following subsections explore the essential elements of AOP specifically.

The manageability and extensibility of AOP is particularly valuable for configurable systems that crosscutting concerns make up a significant part of the whole system. AOP provides an open and generic interface to nonfunctional aspects. Aspect configuration can be controlled and changed at run-time with an immediate effect on desired objects. AOP is also indispensable in supporting new aspects such that it allows implementations of an aspect module to be dynamically replaced in order to fulfill a new aspect configuration.

In the context of mobile middleware, traditional object-oriented programming techniques does not help much in managing nonfunctional properties and crosscutting issues of mobile services that are not confined to a single mobile service, but affect all the services within the current service composition. In particular, the adaptation mechanism for a specific context is not bounded to a single service, but involves the whole service composition. When developing mobile services, we can leverage AOP for weaving the crosscutting issues – *contextual changes* on context adaptation. As shown in figure 4, the concept is to define and supply a rich set of context-independent mobile services that serves different types of functionalities. Then, by using AOP, a mobile application can inject any dimensions of new adaptation behaviors into the mobile services, such that the services serving the mobile application can always adapt precisely to the changing environment.

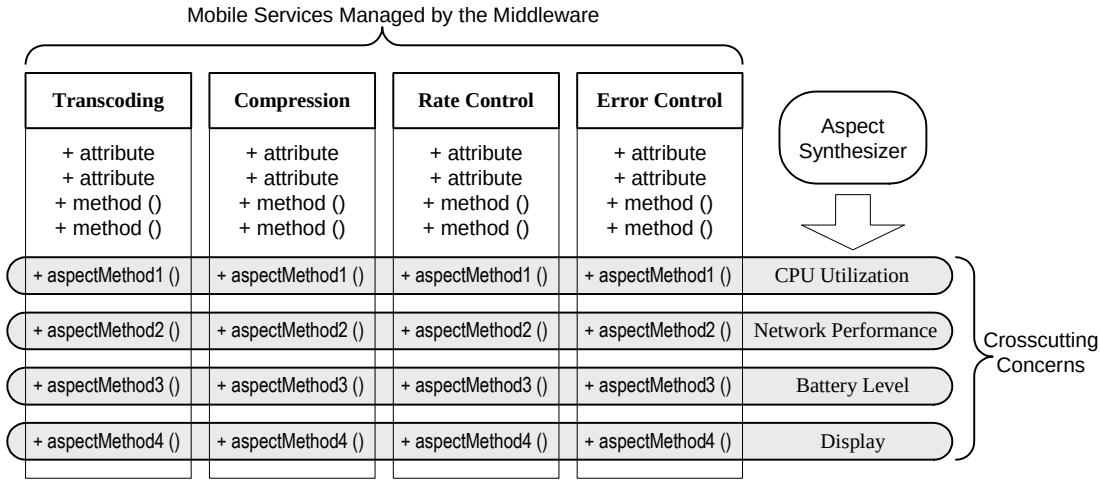

Figure 2.4 Aspect Weaving of Crosscutting Concerns into Context-Independent Mobile Services

2.2.2 Reflective Dynamic Adaptation

Computational reflection [Smith84] is a unique approach to achieving adaptation and re-configuration in a mobile middleware system. In general, computational reflection is a computer process involving self-awareness. Just as with humans, reflection depends on the capacity for independent reasoning, and particularly, reason about one's own processes. A reflective program has the ability to metaprogram [Cordy92] - it can write programs on itself. Specifically, reflection characteristic refers to the ability of a system to monitor its computation and possibly change the semantics of the way it is performed. In other words, a reflective middleware possesses the unique ability to model itself through self-representation, such that manipulation of its behavior may be changed through *introspection* and *interception* [Parlavantzas00]. In this case, introspection refers to the ability of the system to observe and therefore reason about its own state, while interception is the ability of the system to modify its own execution state or its own interpretation or meaning. A middleware system with self-representation is causally connected if changes made to the

self-representation directly affect the implementation of the middleware. The opposite is true if changes to the middleware implementation will change the self-representation.

While the dynamic adaptation of QoS middleware offers some degree of context adaptation, at times mobile applications are still in the best position to make critical decisions on the operating context and hence the adaptation strategy. For this reason, it is desirable for a QoS middleware to provide the mobile application with an extensive set of APIs and reflective interfaces. Through the meta-level object representation of the internal event system and service reconfiguration mechanism, a mobile application can access the contextual information, service configuration and adaptation strategy of the QoS middleware, and modify these entities to obtain optimal service provision from middleware.

2.2.3 Fuzzy Control for QoS Management

Fuzzy control has been successfully applied to various application-specific network QoS management systems [Tsang98] [Pitsillides97] [Chemouil95]. Fuzzy control models [Li99] [Koliver02] have been formulated to address QoS management at high levels of abstraction. These models are limited, however, to abstract modeling and simple conceptual scenarios that do not consider the overall scalability of the model. Specifically, an increase in the number of QoS parameters not only leads to an exponential increase in the number of rules that an area expert must input, it also requires more computational resources to process them. Another concern when applying fuzzy control on mobile QoS management systems is rule reusability. Studies have been done on applying fuzzy control on specific applications, such as video streaming [Tsang98], flow control

[Pitsillides97], and routing [Chemouil95]. However, the sets of fuzzy rules used in these applications were very specific to their corresponding scenarios. We would have to use an entirely different set of fuzzy rules for every new application. Even modifying an existing application requirement to add input or output QoS parameters would require a major revision of the entire set of fuzzy rules. This issue significantly hinders the use of fuzzy control for a mobile QoS management system that is required to support evolving adaptive services and to serve new applications.

2.2.4 Hierarchical Fuzzy Control

Hierarchical fuzzy systems [Raju91] [Ronald98] have been introduced to reduce the number of rules by hierarchically inferring and grouping correlated linguistic variables into abstract linguistic variables for the input of higher-level fuzzy rules. A mobile QoS management system can leverage the hierarchical structuring of fuzzy rules to decouple the interwoven inference of user satisfaction, resource availability, service provision, and adaptation decisions. Each of these categories of parameter forms an independent fuzzy rule hierarchy, so that changes in the parameters in one category do not affect the fuzzy rules of other categories. Using hierarchical fuzzy control also promotes the fuzzy rule reusability of a mobile QoS management system, in which only minimal changes of the corresponding rule hierarchy are required to support new applications, services, and resource parameters.

As an example of hierarchical fuzzy control, assume that there is a CPU intensive mobile application that has a limited battery capacity but must run for a certain period. Adaptation decisions have to be made to balance four QoS parameters.

Assuming that each parameter has five linguistic values, with a plain fuzzy rule structure, the rule pattern is as follows:

IF (CPU_Performance is a_i) AND (CPU_Availability is b_i) AND (Battery_Level is c_i) AND (Power_Conservation is d_i) THEN (action e_i). (1)

This fuzzy system must maintain as many as $5^4 = 625$ rules.

Alternatively, by using hierarchical fuzzy control, the system can be modeled using three rule bases:

IF (Battery_Level is c_i) AND (Power_Conservation is d_i) THEN (Battery_Life is f_i); (2)

IF (CPU_Performance is a_i) AND (CPU_Availability is b_i) THEN (CPU_preferred is g_i); (3)

IF (Battery_Life is f_i) AND (CPU_preferred is g_i) THEN (action e_i). (4)

At most, there will be $5^2 + 5^2 + 5^2 = 75$ rules to be built. As a result, the number of rules to be managed and inferred is greatly reduced by using hierarchical fuzzy control. Notably, hierarchical fuzzy control reduces the maximum number of possible adaptation actions from 5^4 to 5^2. However, considering the significant saving in the number of rules to be managed and inferred, we believe that this drawback is insignificant, as the reduced number of possible adaptation actions is still adequate for mobile QoS management.

Importantly, hierarchical fuzzy control naturally promotes the ease of managing rule reusability by hierarchically grouping related QoS parameters and expressing their relationships through directed graph connections. Changes in user and application needs or even the addition of new QoS parameters only require changes in the corresponding fuzzy rules of the affected rule level, rather than changes to all of the rules as would be required in a flat fuzzy rule structure. For example, if we

have a new application that makes different demands on battery life, only the rules of (4) will have to be modified. Using a flat fuzzy rule structure, all of the rules in (1) would have to be revised.

A potential difficulty of employing hierarchical fuzzy control is the categorizing of different variables, as not all input parameters have clear associations with other input parameters. The inferred abstract linguistic variables must also have physical meanings; otherwise, it would be impossible to build the next level of fuzzy rules using these abstract linguistic variables as the input. Importantly, using hierarchical fuzzy control for mobile QoS management does not cause these problems, since many low-level QoS parameters are closely related, which provides us with the opportunities to model the QoS parameter as a tree structure. Along with a well-designed hierarchy, each inferred abstract linguistic variable can have meaningful representations and is suitable for inferring composite fuzzy rules.

2.3 Summary

In this chapter, we have illustrated the research background for this thesis. Due to the adverse effects bring by the dynamic characteristics of mobile environment, mobile QoS management is essential for mobile application to function efficiently. These dynamic characteristics have various impacts on the design of different QoS processes of a mobile QoS management system. Significantly, the QoS adaption process has become the major concern of Mobile QoS to manage the system and environmental dynamics. We have looked into different adaptive mechanisms to study how these mechanisms can be utilized to support QoS adaptation in mobile environment.

Chapter 3

MobiPADS: A QoS Middleware for Context-aware for Mobile Computing

An important requirement of a middleware system to support mobile computing applications is the provision of a highly configurable and adaptive execution environment that dynamically reacts to changes in operating context. This requirement translates to the need for middleware to organize and implement its system components as a collection of services that are highly configurable and robust enough to enable the system itself to respond to the varying conditions in the environment. In addition, mobile applications are presented with open programming interfaces to enable application introspection and, if required, to re-configure the underlying services to adapt to changes in the environment.

In this chapter, we introduce the **Mobi**le **P**latform for **A**ctively **D**eployable **S**ervice (MobiPADS) system. MobiPADS is designed to support context-aware processing by providing an executing platform to enable re-configuration of the service mix in response to an environment where the context varies. Unlike most mobile middleware, MobiPADS supports dynamic adaptation at both the middleware and application layers to provide flexible configuration of resources to optimize the operations of mobile applications. Within the MobiPADS system, services (known as mobilets) are configured as chained service objects to provide augmented services and protocols to the underlying mobile applications so as to alleviate the adverse conditions of a wireless environment.

3.1 The MobiPADS Framework

Figure 3.1 shows the MobiPADS system architecture. It is composed of two agents: a MobiPADS server at the wired network and a MobiPADS client at a mobile device attached to the Internet through wireless or cellular networks. The two agents marshal the traffic over the wireless link and provide an optimal operating environment for mobile applications. The MobiPADS server is located at or close to the network of the wireless access point, to which the mobile device is connected. The MobiPADS server is designed to support multiple MobiPADS clients and is responsible for most of the optimization computations. The MobiPADS client is an intermediary that provides a comprehensive set of network and system services for mobile applications. These services enable ease of introduction of context-awareness and adaptation for mobile applications, so that the mobile application can adaptively react to varying context environments.

Each MobiPADS agent is composed of two parts: the system components and the MobiPADS service space. The system components provide essential services for the reconfiguration and management of user service pairs – the mobilet pairs, which form the units of service for execution under a MobiPADS environment. The system components also provide common facilities that serve mobilets, which in turn provide value-added services to the wireless environment. These mobilets can be added, updated and removed dynamically.

In the MobiPADS service space, a series of mobilets is linked together to form a processing chain – *the service chain,* which allows mobile applications to benefit from the aggregated functionalities of a collection of mobilets. Mobilets access

the services of the system components though the mobilet API, which also provides interfaces to allow the system components to communicate and configure the mobilets. To monitor the contextual changes, the MobiPADS employs composable event objects that report any contextual change to entities that subscribe to them. The composition of events can be initiated at start-up time, and also allows runtime modification of the event compositions. At the top level of the service space, there is a set of meta-objects that reflects the configuration of the composite events and service chain, as well as the adaptation policies. Both the middleware and the mobile application can use the meta-objects to inspect and reconfigure the event compositions and service chain when adaptation is needed.

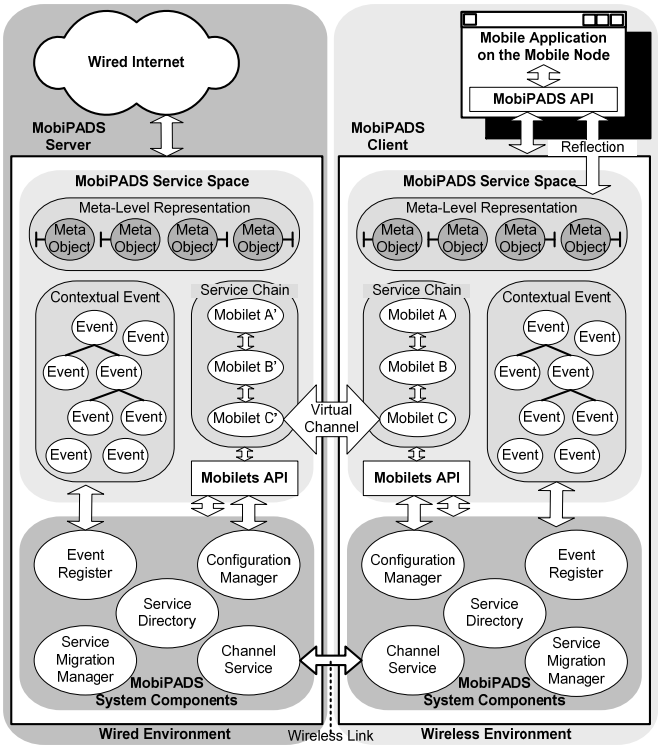

Figure 3.1 The MobiPADS System Architecture

3.2 Mobilet Service Model

A Mobilet is a service entity that can be downloaded, pushed or migrated to a MobiPADS platform for execution within an environment. Mobilets are named after applets, which are active codes executed within Web browsers. Mobilets are active mobile codes that run within the MobiPADS environment.

Mobilets exist in pairs: a master mobilet resides at the MobiPADS client and a slave mobilet resides at the MobiPADS server. A pair of mobilets cooperates to provide a specific service. A typical case would be for a slave mobilet to share the majority of processing burden. A master mobilet instructs the slave mobilet on what actions to take and presents the processed output to the MobiPADS client.

The mobilets are chained together on the client in a specified order, and the corresponding peer mobilets are chained together in a nested order on the server-side. The service chaining model of mobilets supports a general service composition paradigm that enables the utmost flexibility in deploying service aggregation while providing ease of re-configuration in response to the varying characteristics of a wireless environment. A consistent synchronization and data flow model can be established through the abstraction of channel objects and the employment of data encapsulation between services.

In order to support a robust service configuration, all mobilets can be dynamically deployed across a MobiPADS client and server. In other words, it is possible for a mobile node to carry with it relevant mobilets as it travels across foreign domains. As the need arises, mobilets from the client can be dynamically

pushed to a MobiPADS server and configured to operate in a coordinated manner. Conversely, it is possible for a MobiPADS server to push mobilets to a MobiPADS client to actively install new services to operate across a wireless link.

3.3 System Components

The components of the MobiPADS system are briefly described as follows:

Configuration Manager: The configuration manager is responsible for negotiating the connection between the client and the server. It also has a service controller for initializing, interconnecting and managing the mobilets.

Hierarchical Fuzzy Inference Engines: These engines give generalized hierarchical representations – called *QoS factors***,** which captures the overall operating environment and user perceived quality of service. These QoS factors are cascaded and further inferred to decide the importance of QoS factors that are critical to the current application and operating context.

Reconfiguration Engine: The reconfiguration engine matches the importance values of the QoS factors with the mobile profiles, such that the optimal combination of mobilets can be discovered and selected for reconfiguration.

Service Migration Manager: The service migration manager manages the process of importing and exporting mobilets between the MobiPADS server and the MobiPADS client. It also cooperates with the service directory to activate, store and keep track of the changes made to the active mobilets.

Service directory: The service directory records all the known mobilet service types. The object codes are stored in a service repository, which is used for service activation and service migration.

Channel Service: The channel service provides virtual channels which the mobilets use to communicate. Instead of opening separate TCP connections for each message, messages are multiplexed into a single persistent TCP connection, which then eliminates the overheads of opening new TCP connections and avoids the slow-start effect on overall throughput [Liljeberg95].

3.4 Dynamic Service Reconfiguration

To adapt dynamically to changes in an environment, MobiPADS employs the environment monitor and event system to monitor and communicate changes. After changes have been detected, the MobiPADS system can respond in two ways. The first way it can respond is by reconfiguring the current service chain. By adding and removing mobilets within the service chain, the optimum set of mobilets can be selected based on the constrained environment. The second way it can respond is by communicating the changes in the environment to each of the mobilets so that they can readjust their service provision to adapt to the mobile environment.

3.4.1 Service Policies

Figure 3.2 shows the conceptual initialization procedures of the service chain. When a MobiPADS client starts executing, the service controller (of the configuration manager) invokes the profile parser, which will then load and

process the system profile from which the default meta-chain and a number of alternative meta-chains are created. The list of meta-chains is returned to the service controller, which will then deploy the default service chain and the event monitors of the meta-chains. Each meta-chain is attached to an environment monitor, which regulates the time and conditions that determine when the reconfiguration is to take place. When all of the conditions of a specific environment monitor are fulfilled, the corresponding meta-chain will be reflected onto the current service chain, and reconfiguration will take place.

3.4.2 Service Chain Reconfiguration

Service chain reconfiguration takes place when the context environment changes to a state that fulfills all of the conditions of a specific environment monitor. The corresponding meta-chain will then be reflected onto the current service chain to best adapt to the changes. Figure 3.3 shows the procedures of service chain reconfiguration when an environment monitor *EnvMonitor_C* is qualified for reconfiguration. First, the eligible meta-chain will be compared to the active meta-chain (of the current service chain), so that a list of instructions is generated to perform the actual operations needed for reconfiguring the current service chain. As shown in Figure 3.3, the active meta-chain consists of *mNodes_A, mNodes_B,* and *mNodes_C,* while the new meta-chain consists of *mNode_B, mNode_C* and *mNode_D*. The configuration manager compares each mNode in the active meta-chain to the new meta-chain, and any unmatched mNode in the active meta-chain will be marked for deletion. In this case, the *mNode_*A is not found in the new meta chain, thus the instruction *remove(A)* is generated. After all *mNode* in the active meta-chain are compared, a suspension instruction

suspendAll() is added. Then the comparison is repeated but reversed: Each *mNode* in the new meta-chain is compared to the active meta-chain, and any unmatched *mNode* in the new meta-chain will be marked for addition. Subsequently, the instruction *insert(D, 2)* is generated, where the argument *2* is the insert index position in the current service chain. Last, the list of the instructions is passed to the service controller, in which the actual service chain reconfiguration operations are carried out.

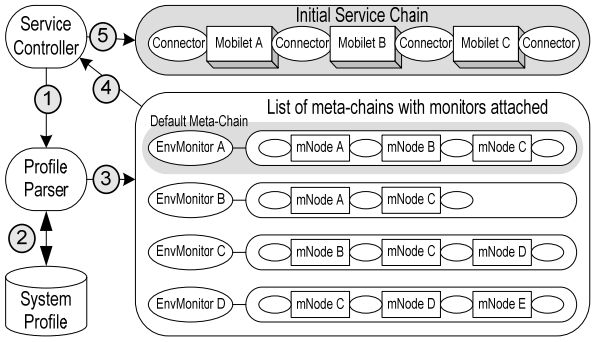

Figure 3.2 Establishment Of The Meta-Chains And Initial Service Chain

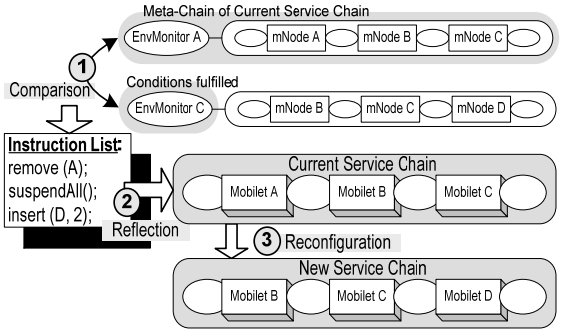

Figure 3.3 Service Reconfiguration Process

3.4.3 Mobilet Reconfiguration

Simply adding or removing mobilets within the service chain is not a sufficiently adaptive response to changes in a wireless environment. To allow a finer-grained adaptation, the MobiPADS programming platform should allow

reconfiguration at the level of individual mobilets. To develop a reconfigurable mobilet, the service object can leverage and dynamically extend the event system and environment monitor. In particular, the mobilet can subscribe to an event and allow it to react to the event messages by adjusting its internal parameters to best adapt to the changes in the environment. If necessary, the mobilet may change the subscription of *EnvMonitor* to adapt to the changing requirements of context monitoring. An example is an image transcoding mobilet that provides different levels for the compression ratio. It can use a different compression ratio based on the reported bandwidth from an event source that monitors the bandwidth. However, it is not desirable to have the mobilet adapt to the contextual changes implicitly. Rather, a set of operation modes should be defined, which will allow external entities to override the adaptation logic of the mobilet, and enforce a specific mode of operation.

3.5 Adaptation Mechanisms

In order to answer the call for multi-dimensional adaptation needs raised by the diversity and dynamics of operating environments, mobile devices, users, applications and usage scenarios (see Table 1), MobiPADS has introduced five layers of adaptation mechanisms that flexibly support adaptation needs at different service layers.

QoS Factor Hierarchy Extension allows the addition of new QoS dimensions to the existing QoS factor hierarchy, such that the number and variety of managed QoS parameters will not be bound by the original design. Importantly, this mechanism is designed to cause minimum disruption to the entire fuzzy rule base

but to update only the direct parent node of the new QoS factor within the hierarchy. This obviates the effort required for a non-hierarchical fuzzy rule base to review the whole rule base when new input is added.

Table 3.1 Five Layers of Adaptation Mechanisms

Mechanism		Objective	Service Layer	Operation
Reflection	**QoS Factor Hierarchy Extension**	Adding new QoS dimensions to the existing QoS Factor hierarchy	Middleware initialization	Adding and changing the fuzzy rules of the direct parent node of the new QoS factor within the hierarchy
	Membership Function Adaptation	Adapting to the specifications on individual QoS factors	Service session initialization during application startup, and run-time response to user's adjustment	Adjusting the membership functions of the linguistic values of the concerned QoS factors
	Importance Weight Adaptation	Adapting to the specifications on overall QoS factor priorities		Changing the importance weights of different concerned QoS factors to reflect the user and application specific priorities
	Fuzzy Control Output	Adapting to the system dynamics and optimizing the QoS perceived by user	Real-time response to the system dynamics	Selecting the optimal set of service profiles
		Allowing applications to access control states and adaptation behaviors through meta-representations	Application initiated real-time meta-level adaptation	Providing Reflective API for applications to inspect (through reflection) and to adapt (through reification) the behavior of the underlying system components, adaptation rules and actions

Membership Function Adaptation allows ad-hoc adaptation to the specifications of a new application. This is done by dynamically adjusting the membership functions of the linguistic values of the involved QoS factors. Traditionally, each set of application specification requires a custom-made set of fuzzy membership functions and fuzzy rule base to support the specific application requirements. However, it is undesirable for an application to specify its own set of membership functions and fuzzy rules, which are too low-level and fuzzy domain specific for typical application developers to handle. The membership function adaptation mechanism is much simpler that it automatically adapts the fuzzy membership functions of concerned QoS factors to the desired formulations specified by user and application.

Importance Weight Adaptation supports ad-hoc user based QoS specification. Based on a similar requirement as in Membership Function Adaptation, this mechanism tries to avoid a redesign of the fuzzy rule base and fuzzy membership functions when a set of new user requirements has emerged. This is done by dynamically changing the importance weights of different QoS factors to reflect the priorities of different QoS factors as specified in the user requirements.

Fuzzy Control Output adapts to the system dynamics and optimizing the QoS perceived by user. This mechanism responds to the system dynamics in real-time by selecting the optimal set of service profiles periodically.

Reflection provides Reflective API for applications to inspect (through reflection) and to adapt (through reification) the behavior of the underlying system components, adaptation rules and actions. While the dynamic adaptation of QoS middleware offers some degree of context adaptation, at times mobile applications are still in the best position to make critical decisions on the operating context and hence the adaptation strategy. For this reason, it is desirable for a middleware to provide the mobile application with an extensive set of APIs and reflective interfaces. Through the meta-level object representation of the internal event system and service reconfiguration mechanism, a mobile application can access the contextual information, service configuration and adaptation strategy of the middleware, and modify these entities to obtain optimal service provision.

Chapter 4 will describe the *extension of QoS factor hierarchy* and *fuzzy control output* and Chapter 5 will present *membership function adaptation*, *rule weight*

adaptation and *reflection*.

3.6 Summary

The growing importance of mobile computing has given rise to a need to re-visit the design requirements of future middleware to cope with the diverse challenges of operating over a dynamic context. The fundamental assumption of a static operating environment, which resulted in a monolithic "black-box" approach to implementing existing middleware, is invalidated in a mobile computing environment. An important requirement in the formulation of a context-aware middleware is the need to devise suitable control mechanisms that allow applications to directly participate in resource adaptation in response to the dynamic operating environment. In this chapter, we have presented the overall architectural design of the MobiPADS system. The MobiPADS represents a reflective-based mobile middleware that is designed to support the dynamic reconfiguration of augmented services for mobile computing. The underlying MobiPADS is implemented as a collection of active-service entities, known as a mobilets, which are constructed as a series of primitive services that form a service-chain composition. The reflective model provides meta-interfaces for applications to directly participate in computation adaptation in response to the changing context. Through the meta-level object representation of the internal event system and service reconfiguration mechanism, a mobile application can access contextual information, the service configuration and adaptation strategy of MobiPADS, and examine and modify these entities to obtain optimal service provision from the MobiPADS.

Chapter 4 Mobile QoS Management Based on Hierarchical Fuzzy Control

The control model of a fuzzy control system is typically modeled by domain experts. Our proposed fuzzy QoS organizes the model into a hierarchical control granularity. This management framework reduces the effort that mobile application developers would otherwise have to put into understanding and controlling every detail of the contextual environment. A mobile application developer using this framework need be aware only of contextual details down to a level sufficient for making adaptation decisions. This obviates the need to manage low-level contextual parameters. The hierarchical fuzzy control model supports a systematic approach that helps application developers easily specify desired adaptation policies. This model can also efficiently, flexibly, and accurately map these application specifications into the adaptation behaviors of the management framework.

4.1 The Hierarchical Fuzzy Control Model

Figure 4.1 provides an overview of the hierarchical fuzzy control model. Unlike typical fuzzy systems, which usually have only a single inference engine, our model is composed of three fuzzy inference engines and one reconfiguration engine, each with its own basic set of fuzzy rules. When there is a large number of QoS parameters, the fuzzy rule hierarchy can greatly reduce the number of fuzzy rules to be managed and inferred, thereby avoiding the rule explosion problem.

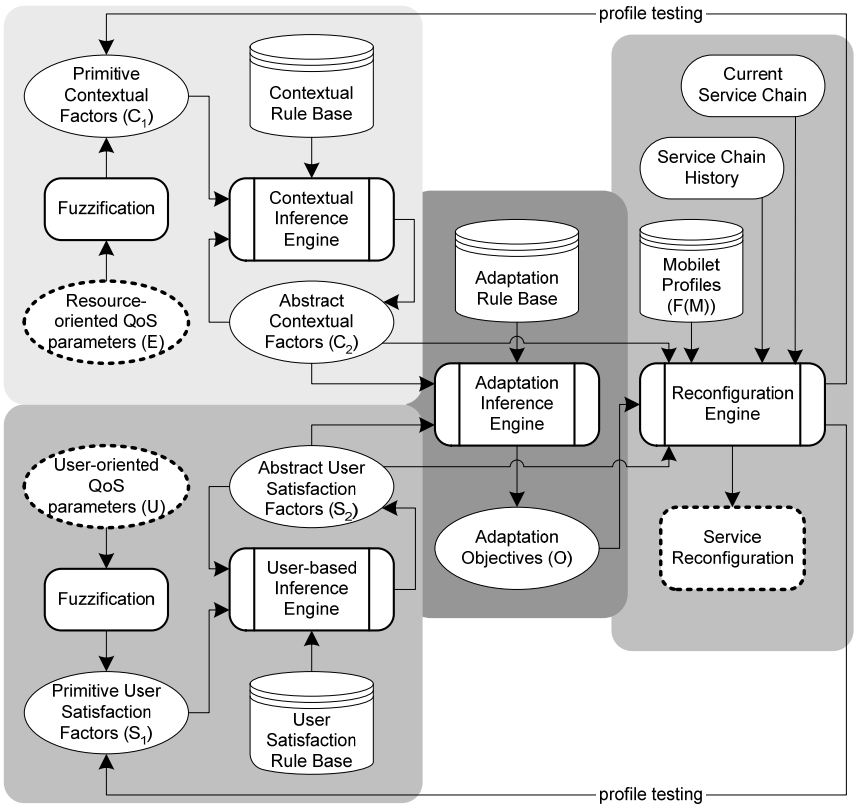

Figure 4.1 Control Flow of the Hierarchical Fuzzy Control Model

The hierarchical fuzzy control model contains four core engines. The *contextual inference engine* in the upper left of Figure 4.1 is responsible for processing and summarizing contextual information. It provides a generalized representation of the overall operating environment in terms of abstract contextual factors such as CPU availability, battery level and network delay. The *user-based inference engine* in the lower left handles the user satisfaction factors, generalizing the overall user satisfaction in terms of abstract user satisfaction factors such as cost of network connection fee, media fidelity and media smoothness. The *adaptation inference engine* in the middle is responsible for inferring a set of adaptation importance values based on the abstract user satisfaction factors and the abstract contextual factors. These importance values describe priorities of different aspects of the middleware services that need adaptation. The *reconfiguration engine* on

the right reacts to the adaptation objectives by reconfiguring the service chain while preserving overall user's satisfaction and resource consumption.

The QoS management framework is designed to support context-aware processing by providing an executing platform to enable mobile services to be actively deployed and reconfigured in response to an environment where the context varies. To alleviate the adverse conditions of a wireless environment, services (called mobilets) are configured as chained service objects (a service chain) to provide augmented services and protocols to the underlying mobile applications. Each mobilet provides a specific functionality. Different combinations of mobilets are chosen to fulfill different QoS requirements and to suit different contextual environments. Service chains can be reconfigured during runtime to adapt to changes in QoS requirements and to the operating environment by adding and removing mobilets. Moreover, each mobilet can have various modes of operation (profiles) to support finer adaptation levels, the details of which are discussed in Section 4.3.2. The mobilet service model enables QoS support for mobile environments, extending from the system resource level up to the user level.

In order to support reusability and scalability, it is essential for the control model to achieve a separation of concerns that would decouple different aspects of the system. This would allow the rules of individual rule bases to be created and updated independently during different phases of development without causing interference between rule bases and without requiring revisions to the rules of other rule bases. New applications are supported through the runtime service adaptation of the QoS factors – namely – the contextual factors and user satisfaction factors.

New mobilets can be added to the system without the need to revise other fuzzy rule bases or the profiles of other mobilets. This design allows the QoS management framework to be flexibly customized to adapt to the characteristics of different mobile devices.

4.2 Hierarchical Inference Engines

This section describes in detail the composition of components and their interactions in the fuzzy control hierarchy. These components drive the inference engines described in Section 4.1.

4.2.1 Resource-oriented QoS Parameters

To support a specific application, a number of system resource QoS parameters are involved. Each represents a measured value of a resource type. The set of resource parameters is denoted by:

$$E = \{e_1, e_2, \ldots, e_p\}, \quad (5)$$

where p is the number of parameters involved. Examples of these parameters might be network delay, CPU utilization, and battery life. To avoid extreme dynamicity of the resource parameters, preprocessing should be applied to control the update frequency, unit conversion, and calibration of the raw measured value. This will ensure that the primitive resource parameters represent normalized values associated with the corresponding aspects. For example, raw CPU utilization measurements can be as precise as the number of occupied CPU cycles within the period of 1 ms. However, it would be more appropriate to normalize this raw measurement to a percentage scale and update it every second, and input this information to the control model. For simplicity, we assume that all parameters in

E of the model have been preprocessed and normalized.

4.2.2 Contextual QoS Factors

Contextual QoS factors are direct or recursive fuzzifications of the resource QoS parameters, which act as input fuzzy variables to the hierarchical fuzzy control model. The set of contextual factors that are relevant to supporting the application is denoted by:

$$C = \{c_1, c_2, \ldots, c_r \mid r \geq p\}. \quad (6)$$

The set **C** consists of two subsets: $\mathbf{C_1}$ and $\mathbf{C_2}$. The subset $\mathbf{C_1}$ holds *primitive contextual QoS factors* and it is denoted by:

$$C_1 = \{c_1, c_2, \ldots, c_p\}, \quad (7)$$

whose elements $\boldsymbol{c_i}$ are a direct fuzzification of the corresponding $\boldsymbol{e_i}$ in **E**. The subset $\mathbf{C_2}$ is comprised of *abstract contextual QoS factors* and it is denoted by:

$$C_2 = \{c_{p+1}, c_{p+2}, \ldots, c_r\}, \quad (8)$$

which are generated from the nested fuzzification of elements in C_1. C_2 represents high-level descriptions of the contextual environment. For example, “battery life is good” or “network performance is poor.”

As shown in Figure 4.2, we organize contextual information of different levels of abstraction into a tree hierarchy. Fuzzy rules in the contextual rule base can be defined using both primitive contextual factors and abstract contextual factors as input linguistic variables. For example,

```
IF(Battery_Level is normal) AND (Power_Conservation is good) THEN (Battery_Life is good);
IF (CPU_Availability is normal) AND (Battery_Life is poor) THEN (CPU_Preferred is poor).
```

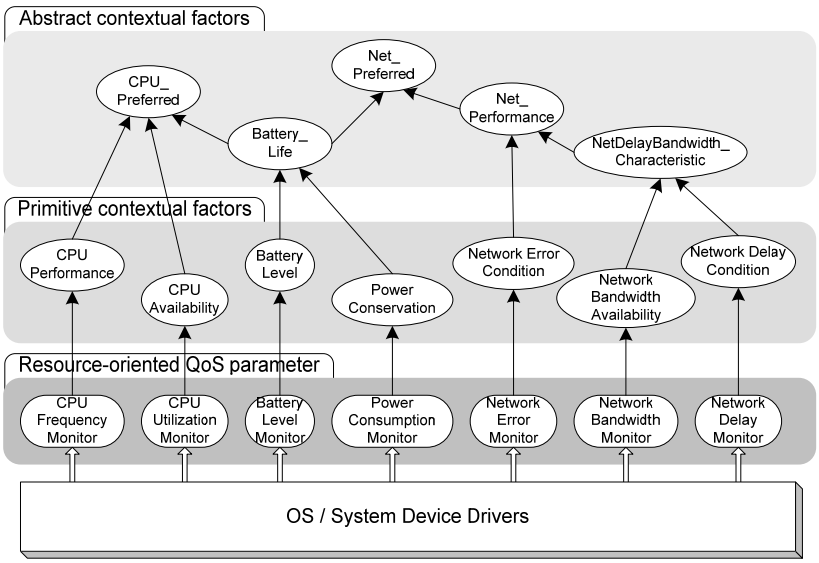

Figure 4.2 Contextual QoS Factor Hierarchy

4.2.3 User-oriented QoS Parameters

Typical mobile application users are not concerned about the low-level resource parameters or the contextual factors. They are concerned with what they can perceive about the performance of the application in terms of quantifiable quality of service, for example, responsiveness, smoothness, and clarity. To define these abstract concepts, however, we need to obtain concrete user-oriented measurable QoS parameters. The set of concerned user-oriented QoS parameters in a specific application is denoted by:

$$U = \{u_1, u_2, \ldots, u_q\}. \quad (9)$$

Each parameter represents a measured value of a primitive user-oriented QoS parameter, where q is the number of parameters. Examples are resolution, frame rate, and air-time charge per minute.

4.2.4 User satisfaction QoS Factors

Like contextual QoS factors, user's satisfaction QoS factors are the fuzzy variables

that directly or recursively fuzzify user-oriented QoS parameters, as shown in Figure 4.3. The set of user satisfaction QoS factors is denoted by:

$S = \{s_1, s_2, \ldots, s_t \mid t \geq q\}$. (10)

Similarly, $\mathbf{S} = \{\mathbf{S_1}, \mathbf{S_2}\}$. The subset $\mathbf{S_1}$ is defined as *primitive user satisfaction QoS factors* that are the fuzzification of the corresponding s_i in **U**, which is denoted by:

$S_1 = \{s_1, s_2, \ldots, s_q\}$. (11)

The subset $\mathbf{S_2}$ consists of *abstract user satisfaction QoS factors* that represent the high-level user satisfaction factors. It is denoted by:

$S_2 = \{s_{q+1}, s_{q+2}, \ldots, s_t\}$. (12)

Examples of abstract user satisfaction QoS factors might be media presentation quality and application availability. Like the contextual rule base, the user satisfaction rule base contains fuzzy rules that are defined with the user satisfaction factors of both S_1 and S_2. For example,

```
IF (Transmission_Responsiveness is poor) THEN (Responsiveness is poor);

IF (Synchronization is normal) AND (Fidelity is good) AND (Smoothness is normal) AND
(Responsiveness is good) THEN (Quality is good).
```

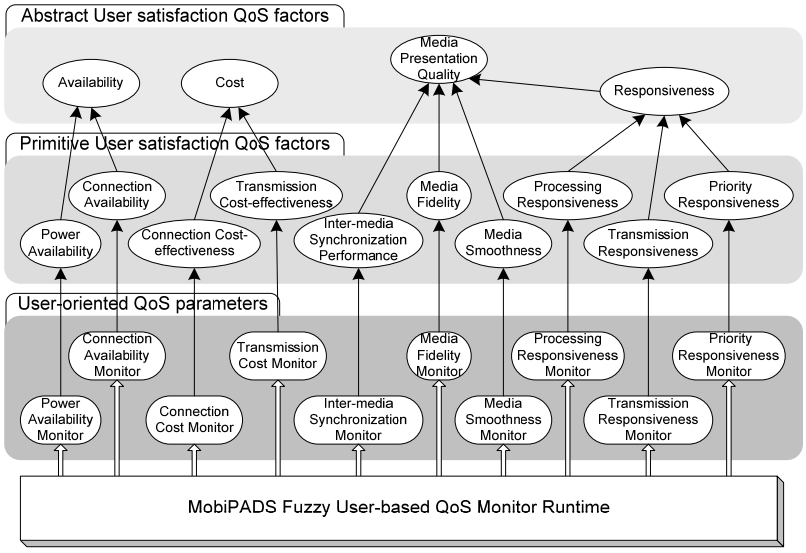

Figure 4.3 User Satisfaction QoS Factor Hierarchy

4.2.5 Adaptation Importance

The adaptation inference engine will make use of the user satisfaction factors and contextual factors to deduce a set of *adaptation importance values* for the next service reconfiguration cycle. The fuzzy rules in the adaptation rule base seek to determine the user satisfaction factors that may need the most improvement, and the extent to which the user satisfaction factors should be improved given the current contextual situation. Such decision output is represented by a list of relative importance values of each QoS factor, whose usage for adaptation will be further discussed in Section 4.4.2 and exemplified in Table 4.2 of Section 4.5.

4.3 Mobilets

In the QoS management framework, a mobilet is a service object that performs a single task. A number of mobilets are selected to form a service chain that provides an integrated set of services to the mobile application. Each mobilet can have a number of operational modes, denoted by mobilet service profiles, which clearly describe the usages, the usable situations, and the impact of the each of the operational modes.

4.3.1 Mobilet Service

Let $\mathbf{M} = \{m_1, m_2, \ldots, m_v\}$ be the set of mobilet services that can be used to support the application, where v is the total number of mobilets in the set.

A mobilet is characterized by a collection of three lists: input types, output types, and service profiles:

$$m = \{\mathrm{IN}(m), \mathrm{OUT}(m), \mathrm{F}(m)\}. \quad (13)$$

The input and output types are used for matching the object types to support the application requirements. Each mobilet can support multiple input and output types. For example, a mobilet accepts raw AVI video format as input and produces H.264 video as output. A mobilet chain template is generated to specify the mapping of compatible mobilets to be selected and chained, while ensuring that input and output types between the connecting mobilets are fully compatible.

4.3.2 Mobilet Service Profile

A mobilet service profile represents a mode of operation that the mobilet can support, e.g. a compression mobilet can have a maximum speed profile, a maximum compression profile, and a balanced profile. The set of profiles of the mobilet m_i can be denoted by:

$$F(m_i) = \{f_i^1, f_i^2, \ldots, f_i^{h(i)} \mid i \in [1,v]\}, \quad (14)$$

where $h(i)$ is the total number of profiles of m_i. The reconfiguration engine selects a list of profiles from **F(M)** as its output, which optimizes the QoS and resource usage under the current operating context. The output of the reconfiguration engine is denoted by:

$$F_{optimize}(M) = \{f_\alpha^a, f_\beta^b, \ldots, f_\delta^d \mid 1 \le \alpha < \beta < \delta \le v,\ a \in [1,h(\alpha)],\ b \in [1,h(\beta)],\ d \in [1,h(\delta)]\}. \quad (15)$$

A profile is denoted by

$$f = \{A(f)\}, \quad (16)$$

where **A(*f*)** denotes a list of QoS factors that will be affected if the profile is activated.

4.3.3 Affected QoS Factor

An affected QoS factor is a three-element tuple: the *parameter label*, *adjustment*,

and *modifier*, which is denoted by:

$A_i(f) = \{c_i, \alpha, x, \mid c_i \in [C_1 \cup S_1]\ , \alpha \in [-1,1], x \in \{0,1\}\}$. (17)

Each tuple describes the effect on a specific QoS factor when the profile is activated. c_i is a primitive contextual QoS factor or a primitive user satisfaction QoS factor. α is the modification value and x is the modifier, which can be either 0 or 1, indicating whether the corresponding adjustment α is either absolute or relative. As an example, if a QoS factor, smoothness, is currently defuzzified to a crisp value of 0.5, then the tuple {smoothness, -0.2, 1} implies that the new smoothness value is (-0.2) + (0.5)(1) = 0.3. In other words, if the profile is activated, the smoothness factor will be affected and changed to a crisp value of 0.3. With reference to (6) and (10), the list of affected QoS factors **A(*f*)** is defined as

$A(f) = \{A_1(f), A_2(f), \ldots, A_k(f) \mid 1 \le k \le p+q\ \}$. (18)

As an example of a complete mobilet service profile, one of the profiles for an MPEG4 encoding mobilet can be

$f_{MPEG4}^{High\_Quality}$ = {`Power_conservation,-0.2,1, CPU_Availability,-0.2,1, Net_Bandwidth_Availability,0.2,1, Fidelity,0.8,0`}.

The adjustment of an activated profile to a QoS factor can be defined in two ways: deterministic and probed. *deterministic adjustment* represents an absolute casting of the specific QoS factor, which is enforced by the mobilet. For example, a video transcoding mobilet enforces its video output to be in QVGA resolution, thus restricting the media fidelity factor to a specific value.

A mobilet can intentionally or unintentionally affect a QoS factor so that the extent of the effect is uncertain during the design phase, in which case probing during the testing and deployment phase is required to determine the actual effect. This is

called *probed adjustment*. Due to its nondeterministic characteristic, probed adjustment is best characterized based on relative representations. For example, a text compression mobilet can reduce HTTP traffic by 30% on average, but as a side effect, it also increases the CPU loading by 10% on average. These relative changes must be probed since they cannot be determined during the design phase. A developer of a mobilet with probed adjustments should implement active testing functions that can be invoked when the mobilet is first installed, to determine the initial values of the probed adjustments. Passive probing functions should also be implemented, which can be invoked when the mobilet is activated for adaptation. This calibrates and updates the values of the probed adjustments adaptively, so that the values can accurately reflect the adjustments to the current operating platform and environment. The probing of service performance [Cherkasova02] [Obraczka98] is a research topic on its own, yet for simplicity of illustration we again assume that the probed adjustments for every profile are accurate and up-to-date.

4.4 Fuzzy-controlled Service Reconfiguration

A large variety of algorithms and techniques [Al-bar99] [Chalmers99] have been developed to provide mobile-enhanced services. Typically, these services have been designed to optimize operations in restricted contexts and for specific applications. Hence, the selection of services becomes a process that is crucial to the efficiency of a mobile operating environment. In this section, we describe a generalized scheme for mobile service selection that uses fuzzy sets.

Given a contextual environment and a set of measured QoS factors, the middleware

have to select the mobilet combination and the mobilet profile that will optimize the QoS factors of specific applications and maintain the optimal level of user satisfaction.

The set of *optimization objectives* represents all of the contextual QoS factors and user satisfaction factors, which can comprise of both primitive and abstract factors as in (6) and (10),

$$O_{optimization}\ (C_1 \cup S_1) = \{o_1, o_2, \ldots, o_k \mid k=p+q\}, \text{ where } o_i = \begin{cases} O_{optimization}(c_i), & i \le p \\ O_{optimization}(s_{i-p}), & i > p \end{cases}. \quad (19)$$

Each QoS factor is mapped to an objective. The *decision function* **G**, is defined by the intersection (to find the minimum) of all optimization objectives

$$G = o_1 \cap o_2 \cap \ldots \cap o_k. \quad (20)$$

A decision scoring of an objective for a profile is determined by the *relative importance* of the objective and the *degree of satisfaction* achieved by the profile.

Although the QoS management framework can support the simultaneous reconfiguration of multiple mobilets in a service chain, the control model limits the change to one mobilet per reconfiguration. This is to facilitate the probing and updating of the affected QoS factors of each mobilet profile. This in turn leads to a simpler design for mobilet selection. In order to select the optimal mobilet profile, we adopt a decision calculus introduced in [Yager81], which is based on ordinal information, as the input for weightings of importance. There are a large variety of fuzzy methods [Fuller96] for multiple-criteria decision making. The decision calculus of Yager [Yager81] is the most suitable method for selecting mobilet profiles because of the simplicity of its model and low computational complexity. More importantly, this method supports our aim of identifying profiles with the

least amount of compromise in the concerned QoS factors, rather than selecting profiles with uneven performances among the concerned QoS factors. This is vital to our system, since a high score in a specific QoS factor is of little value when it reaches a certain level, e.g. a video of 60 frames per second (fps) is indistinguishable in most cases from a video of 30fps. On the other hand, small improvements in any disadvantaged QoS factor can improve user perceived quality notably. In short, the system optimizes the overall performance by improving the worst performing QoS factor one at a time.

Let $\delta(f_j, o_i)$ be the *decision score* of a profile f_j in objective o_i; then, the set of decision scores for f_j is denoted by:

$$D(f_j) = \{\delta(f_j, o_1), \delta(f_j, o_2), \ldots, \delta(f_j, o_k)\} \rightarrow [0, 1]. \quad (21)$$

The *overall decision score* of f_j is then denoted by:

$$\delta(f_j, G) = \min[D(f_j)], \quad (22)$$

where the *optimal profile* f^* is denoted by:

$$\delta_{\text{optimal}}(f^*) = \max_{f \in F(M)} [\delta(f, G)]. \quad (23)$$

As described in section 3.2.5, the importance of an objective o_i is denoted by w_i, which can be determined by the adaptation inference engine. The set of *importance values* of corresponding objectives in $\mathbf{O_{optimization}}$ $\mathbf{(C_1 \cup S_1)}$ is denoted by:

$$W = \{w_1, w_2, \ldots, w_k\} \rightarrow [0, 1]. \quad (24)$$

The degree of satisfaction of objective o_i for a profile f is denoted by $o_i(f)$ and can be computed through *profile testing*. As shown in the top and bottom part of Figure 4.1, the reconfiguration engine tests the profile f by affixing all of its affected QoS factors onto the primitive QoS factors that will subsequently reflect the changes

onto the abstract QoS factors. The testing measurements of the primitive and abstract QoS factors are then taken as degrees of satisfaction.

Let **N(**o_i, w_i**)** denote the new *decision score* of the objective o_i that has an importance of w_i. The decision function in (19) becomes

$$G = N(o_1, w_1) \cap N(o_2, w_2) \cap \ldots \cap N(o_k, w_k). \quad (25)$$

By using the Kleene-Dienes implication operator [Park92], where $I_{KD}(x, y) = \max(1\text{-}x, y)$, the decision score of an objective o_i for a mobilet profile f_j is defined as

$$\delta(f_j, o_i) = N(o_i(f_j), w_i) = w_i \rightarrow \delta(f_j, o_i) = o_i(f_j) \vee \overline{w_i}, \text{ where } \overline{w_i} = (1\text{-}w_i). \quad (26)$$

Figure 4.4 illustrates how the implication operation assigns a score to each of the objectives based on their importance and satisfaction. The lowest score represents the bottleneck of a profile, which is the most representative QoS performance measurement of the profile.

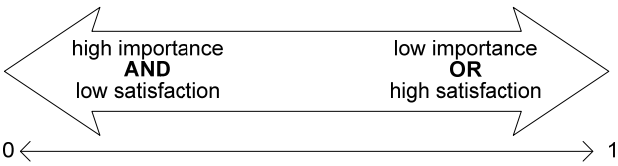

Figure 4.4 Sorted Decision Scores Using The Implication Operator

Based on (25) and (26), the decision function for finding an overall decision score for all objectives becomes

$$G = (o_1 \cup \overline{w_1}) \cap (o_2 \cup \overline{w_2}) \cap \ldots \cap (o_k \cup \overline{w_k}). \quad (27)$$

Then, the *optimal profile* can be found by expanding (23):

$$\delta_{\text{optimal}}(f^*) = \max_{f \in F(M)} \{ \min_{i=1}^{k} [\max(o_i(f), \overline{w_i})] \}. \quad (28)$$

If two conflicting profiles have the same overall decision score, their second-smallest decision score representing their second bottleneck, can be compared. If a conflict persists, the comparison can be repeated.

4.5 Performance Results and Analysis

To test the performance and functionality of the system, we have developed an emulated wireless video streaming application based on the implementation of the QoS management framework. This simulation allows us to study the performance of a generic wireless application running on top of the QoS management framework. The video streaming application exercises and adapts to multiple QoS parameters, including network bandwidth, network error rate, video smoothness, video fidelity, and video noise. The video application also incorporates the coexistence of multiple adaptation options, including the frame rate, resolution, and codec selection. The actual adaptation decisions are made based on the current wireless environment and pre-assigned application preferences, and are assisted by the hierarchical fuzzy control model.

4.5.1 Experimental Setup

Figure 4.5 shows the logical flow of the experimental setup. At the top of Figure 4.5, the data flow of the wireless video streaming application begins with the streaming of a video into a service chain that is composed of a number of mobile-enhanced mobilets. There are three tiers of mobilets along the service chain: the frame rate adaptation mobilet, the resolution adaptation mobilet, and the codec mobilets. These mobilets are dynamically reconfigurable to best adapt to the current contextual environment. They do this by appropriately transcoding

the streamed video into an optimized form to be transmitted over the emulated wireless network. There is a video receiver at the other end of the emulated network, which measures the quality of the received video stream and feeds these QoS measurements back to the QoS monitors.

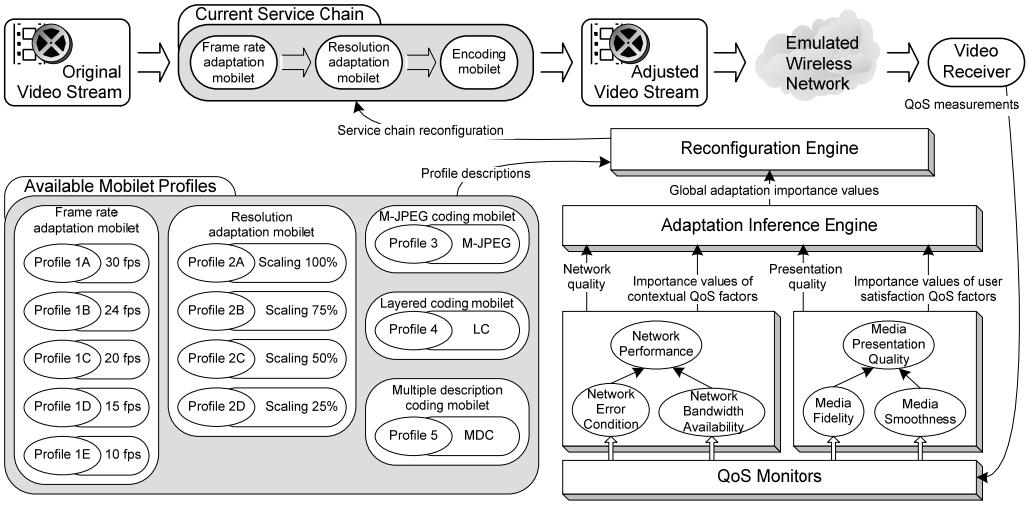

Figure 4.5 An Adaptive Mobile Video Streaming Application

The lower left of Figure 4.5 shows twelve mobilet profiles that can be selected from five mobilet profile sets. The frame rate adaptation mobilet has five profiles: 30, 24, 20, 15, and 10 frames per second. The resolution adaptation mobilet has four resolution scaling profiles: 100%, 75%, 50%, and 25%. There are three codec mobilets, each with just one profile: ordinary M-JPEG, layered coding (LC) [Li97], and multiple description coding (MDC) [Goyal01]. LC has been developed for scalable video delivery, in which the signal is separated into components of varying levels of detail. MDC breaks the data into several streams with some redundancies between the streams. When partial streams are received, the quality of the reconstruction degrades gracefully. In this simulation, we adopt the characteristics and measurements of LC (with ARQ on the base layer) and MDC from [Singh00]. These profile descriptions are submitted to the reconfiguration

engine for a decision as to what profile should be selected. This is done dynamically to reflect the projected effects of the profiles in the existing contextual situation.

The middle of Figure 4.5 shows the control flow of the simulation. The QoS monitors gather the QoS measurements from the operating environment and the video receiver. These measurements are then normalized and input into correspondingly to the contextual inference engine and user-based inference engine. These two inference engines infer the local importance values of their QoS factors, and also the overall quality of their QoS factor hierarchy. The adaptation inference engine then infers the global adaptation importance values for each QoS factor by making use of these measurements of overall quality and local importance values. Once there is a set of global adaptation importance values for each QoS factor and the set of profiles describing the effect of the profiles on each of the QoS factors, it becomes a straightforward task for the reconfiguration engine to decide which profile is the best for the current contextual situation. After an optimal profile set is selected, the reconfiguration engine reconfigures the current service chain to provide an optimal transcoding service for the video streaming application. The whole adaptation control flow is carried out dynamically and periodically so that the current service chain can always match the QoS demands of the application in a dramatically changing wireless operating environment.

We have carried out a series of experiments with the wireless video streaming application. The detailed environmental setup is as follows: A Motion-JPEG encoded video stream with a frame size of 640x480 pixels and a speed of 30 frames

per second is streamed over a wireless network, from a fixed-network server to a wireless client. The QoS management framework is running on both the server and the client, providing transcoding and adaptation services to the video streaming application. The maximum bandwidth of the wireless network is 150% of the video stream, the average round trip time (RTT) of the wireless network is 600 ms, and the average error rate is 10%. At the transport layer, the User Datagram Protocol (UDP) is used for the packet transfer. The packet size is fixed at 100 bytes to minimize the adverse effect due to lost packets. The maximum allowed packet delay is 2000 ms, and each lost packet can be retransmitted only once at most, to prevent unbounded delays.

4.5.2 The Fuzzy Rules

To give a concise illustration of the model so that it can be better appreciated, the number of QoS factors involved in the hierarchical fuzzy control is trimmed down to six: fidelity, smoothness, presentation quality, bandwidth, error rate, and network quality. Table 4.1 shows the fuzzy rules of the network quality rule base. The contextual inference engine uses these rules to generate the relative importance of bandwidth and error rate, and also an abstract QoS factor – *network quality*. Figure 4.6 shows the function surface plot for network quality. We use symmetric Gaussian membership functions for all of the fuzzy rules in this simulation. The user satisfaction rule base and the user-based inference engine work in a similar manner, generating the importance of fidelity and smoothness. An abstract QoS factor – *presentation quality***,** is also inferred by the user-based inference engine.

The relative importance values generated by the user-based inference engine and the contextual inference engine are only valid within their corresponding scopes.

However, we need a global set of importance values to allow quantitative comparisons among all QoS objectives. This is provided by the adaptation inference engine. As shown in Table 2, network priority (netPriority), representing the relative weightings of the contextual QoS factors compared to the user satisfaction QoS factors, is inferred using presentation quality (userQoS) and network quality (netQoS). The function surface plot of network priority is shown in Figure 4.7.

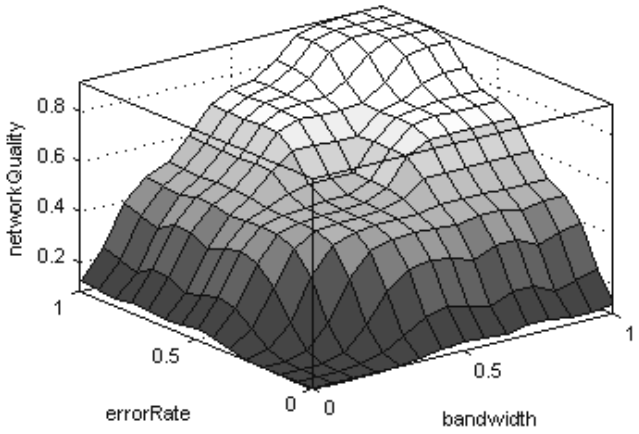

Figure 4.6 Network Quality Function Inferred by Bandwidth and Error Rate

Table 4.1 The Network Quality Rule Base

1. If (bandwidth is excellent) and (errorRate is excellent) then (networkQuality is excellent)
2. If (bandwidth is excellent) and (errorRate is good) then (networkQuality is excellent)
3. If (bandwidth is good) and (errorRate is excellent) then (networkQuality is excellent)
4. If (bandwidth is normal) and (errorRate is good) then (networkQuality is good)
5. If (bandwidth is good) and (errorRate is normal) then (networkQuality is good)
6. If (bandwidth is normal) and (errorRate is normal) then (networkQuality is normal)
7. If (bandwidth is poor) and (errorRate is normal) then (networkQuality is normal)
8. If (bandwidth is normal) and (errorRate is poor) then (networkQuality s normal)
9. If (errorRate is bad) then (networkQuality is bad)
10. If (bandwidth is bad) then (networkQuality is bad)
11. If (bandwidth is excellent) and (errorRate is normal) then (networkQuality is good)
12. If (bandwidth is excellent) and (errorRate is poor) then (networkQuality is normal)
13. If (bandwidth is good) and (errorRate is poor) then (networkQuality is normal)
14. If (bandwidth is normal) and (errorRate is excellent) then (networkQuality is good)
15. If (bandwidth is poor) and (errorRate is excellent) then (networkQuality is normal)
16. If (bandwidth is poor) and (errorRate is good) then (networkQuality is normal)
17. If (bandwidth is bad) then (bandwidthPriority is veryHigh)
18. If (bandwidth is poor) then (bandwidthPriority is high)
19. If (bandwidth is normal) then (bandwidthPriority is medium)
20. If (bandwidth is good) then (bandwidthPriority is low)
21. If (bandwidth is excellent) then (bandwidthPriority is veryLow)
22. If (errorRate is bad) then (errorRatePriority is veryHigh)

23. If (errorRate is poor) then (errorRatePriority is high)
24. If (errorRate is normal) then (errorRatePriority is medium)
25. If (errorRate is good) then (errorRatePriority is low)
26. If (errorRate is excellent) then (errorRatePriority is veryLow)

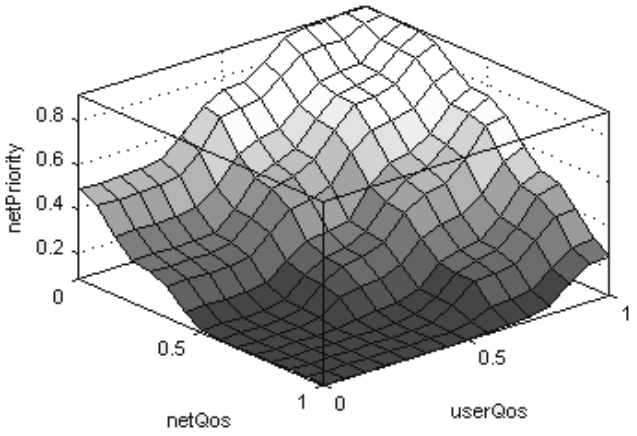

Figure 4.7 Network Priority Inferred by Network and Presentation Quality

Table 4.2 The Adaptation Rule Base

1. If (netQos is bad) and (userQos is bad) then (netPriority is medium)
2. If (netQos is bad) and (userQos is poor) then (netPriority is medium)
3. If (netQos is bad) and (userQos is normal) then (netPriority is high)
4. If (netQos is bad) and (userQos is good) then (netPriority is veryHigh)
5. If (netQos is bad) and (userQos is excellent) then (netPriority is veryHigh)
6. If (netQos is poor) and (userQos is bad) then (netPriority is low)
7. If (netQos is poor) and (userQos is poor) then (netPriority is low)
8. If (netQos is poor) and (userQos is normal) then (netPriority is medium)
9. If (netQos is poor) and (userQos is good) then (netPriority is high)
10. If (netQos is poor) and (userQos is excellent) then (netPriority is veryHigh)
11. If (netQos is normal) and (userQos is bad) then (netPriority is veryLow)
12. If (netQos is normal) and (userQos is poor) then (netPriority is veryLow)
13. If (netQos is normal) and (userQos is normal) then (netPriority is low)
14. If (netQos is normal) and (userQos is good) then (netPriority is medium)
15. If (netQos is normal) and (userQos is excellent) then (netPriority is high)
16. If (netQos is good) and (userQos is bad) then (netPriority is veryLow)
17. If (netQos is good) and (userQos is poor) then (netPriorty is veryLow)
18. If (netQos is good) and (userQos is normal) then (netPriority is veryLow)
19. If (netQos is good) and (userQos is good) then (netPriority is low)
20. If (netQos is good) and (userQos is excellent) then (netPriority is medium)
21. If (netQos is excellent) and (userQos is bad) then (netPriority is veryLow)
22. If (netQos is excellent) and (userQos is poor) then (netPriority is veryLow)
23. If (netQos is excellent) and (userQos is normal) then (netPriority is veryLow)
24. If (netQos is excellent) and (userQos is good) then (netPriority is veryLow)
25. If (netQos is excellent) and (userQos is excellent) then (netPriority is low)

4.5.3 Experimental Results

Two scenarios are tested in this simulation. The first scenario uses a static service chain without frame rate adaptation and resolution adaptation, while using LC as the codec. The second scenario allows free adaptation on all three tiers of mobilets. In these two scenarios, we varied the bandwidth from zero to 400 Kilobyte/s (KB/s), and repeated the scenarios under different bit error rates of between 5% to 40%. Figure 4.8 shows the results that capture the clarity performance of both scenarios, while Figure 4.9 shows the smoothness performance and Figure 4.10 shows the packet drop rate. The plot line labeled *static 0.05* refers to the results for a static-service-chain under a bit error rate of 5%, while the plot lines labeled *dynamic* refer to the results for free-adaptation scenarios.

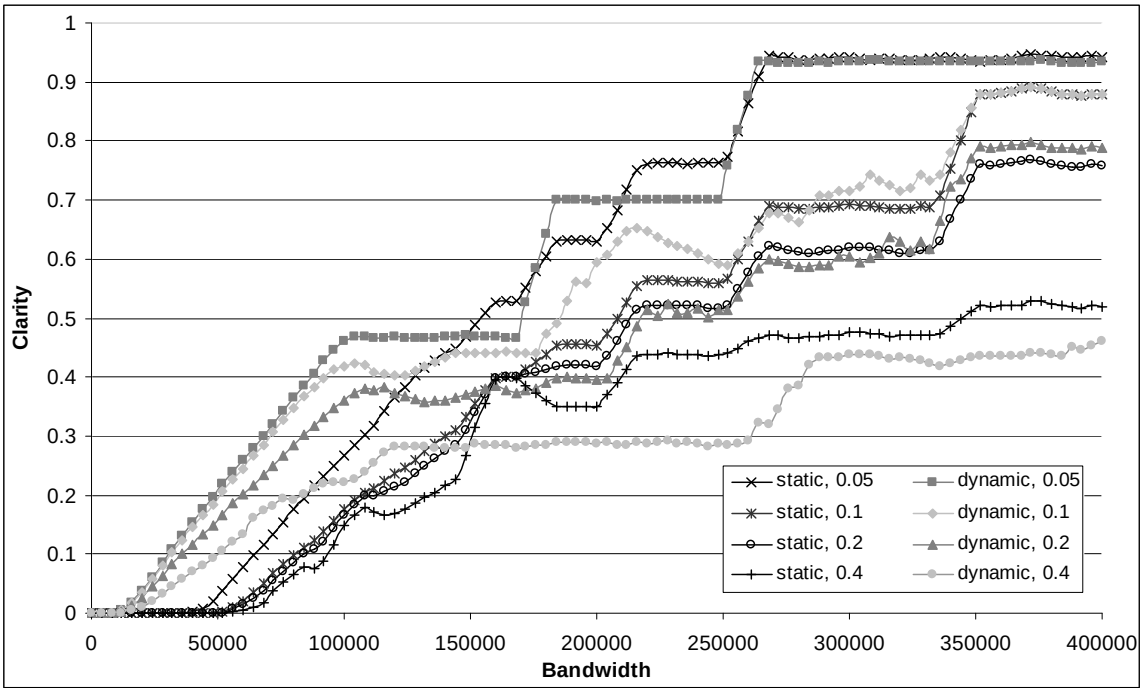

Figure 4.8 Clarity Performance of Static Services Versus Fuzzy QoS Adapted Services

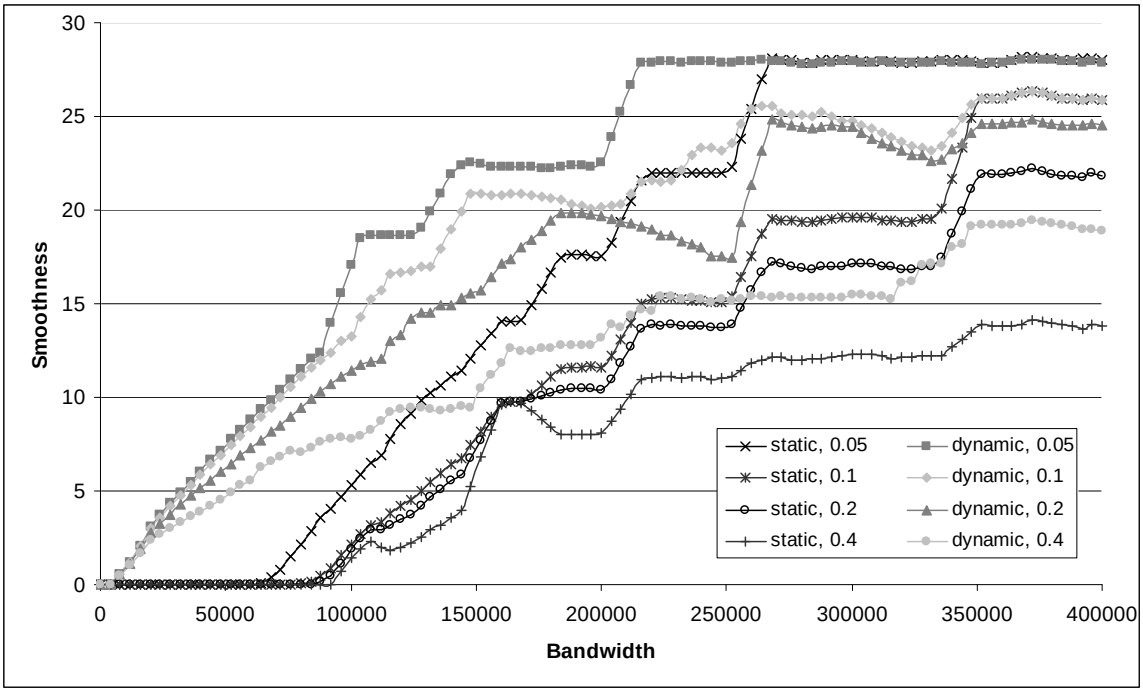

Figure 4.9 Smoothness Performance of Static Services Versus Fuzzy QoS Adapted Services

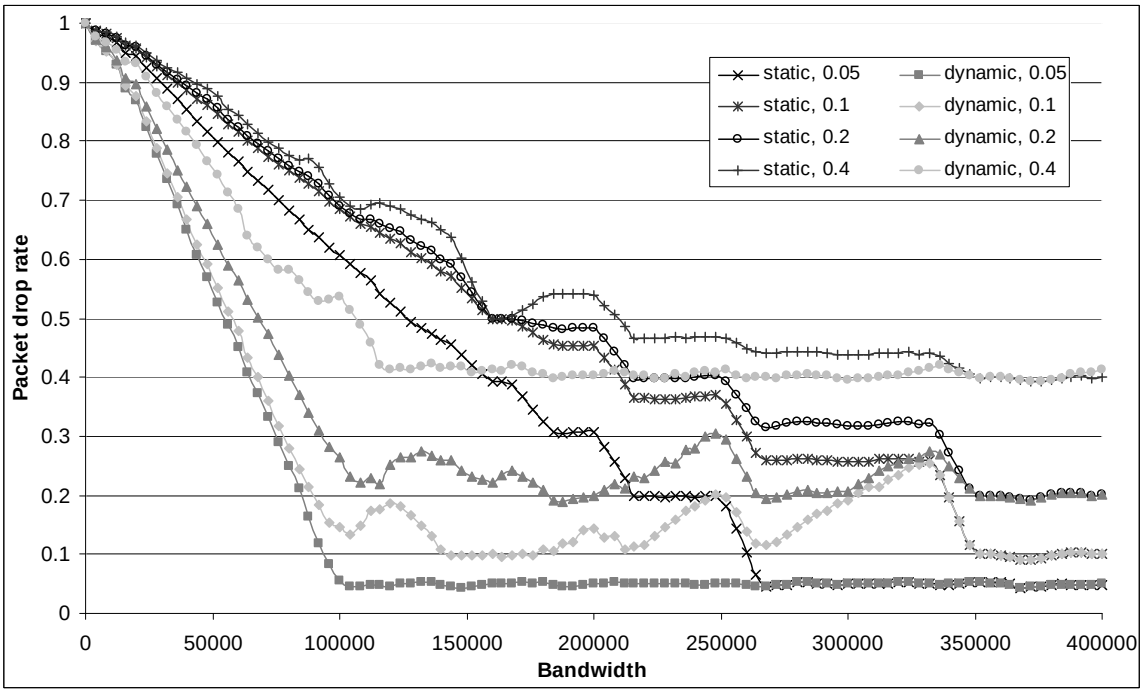

Figure 4.10 Packet Drop Rate of Static Services Versus Fuzzy QoS Adapted Services

Figure 4.8 shows the results capturing the performance of both scenarios in terms of

clarity. It can be seen that free-adaptation results in significantly better clarity performances when the bandwidth is below 160 KB/s, ranging from 7% to 15%. The measurement is captured only for frames that are successfully transmitted and decoded over the wireless link. In other words, dropped frames do not affect the clarity measurement. Higher bandwidth conditions show similar clarity performances for both free-adaptation and the static-service-chain. However, when the bandwidth and bit error are both high, the static-service-chain is 9% better than free-adaptation in terms of clarity. The main reason for this result is that under the free adaptation scenario, the system aggressively reconfigures the resolution adaptation mobilet to adapt to a high bit error rate, which leads to a more conservative consumption of bandwidth and hence to a better overall quality of presentation. The effect of the aggressive adaptation of resolution contributes partially to the results shown in Figure 4.9.

Figure 4.9 shows the performance in terms of smoothness. Under high bandwidth and a high bit error rate, free-adaptation offers a 60% higher frame rate than a static-service-chain. In general, free-adaptation offers significantly better smoothness performances than the static-service-chain. Under various bit error rates, free adaptation achieves a markedly better frame rate than does a static-service-chain, by 4.5 frame/s to 6.6 frame/s. Together, Figure 4.8 and Figure 4.9, show that free-adaptation consistently offers a more balanced performance with regard to clarity and smoothness. Moreover, under various network conditions, the individual performances of free-adaptation are usually on par with or better than those of a static-service-chain.

Figure 4.10 shows the packet drop rates for the two scenarios. The packet drop rate is directly affected by the bit error rate, the bandwidth, and the data rate. As the bit error rate and bandwidth are controlled in the experiments, the packet drop rate is the result of variations in the data rate:

*when data rate > bandwidth * (1-bit error rate),*

then the packet drop rate > bit error rate;

*when data rate < bandwidth * (1-bit error rate),*

then the packet drop rate = bit error rate.

Figure 4.10 further explains the benefits of free-adaptation. Under various network conditions, the system is able to actively control the consumption of bandwidth by reducing the frame rate and downscaling the transmitted video. This prevents the channel from becoming congested when the bandwidth is low. In contrast, the static-service-chain consistently creates congestions within the network due to over-utilization of the available bandwidth, which results in significantly higher packet drop rates of between 9% to 22% compared to free-adaptation These high packet drop rates have a direct impact on the clarity and smoothness of the static-service-chain. In short, the results have clearly demonstrated the benefits of the QoS management framework in mapping and adapting to variations in QoS parameters under varying contextual environments. The experiments have also served to verify the operations of the framework and to provide us with the opportunity to investigate complex interactions among the components within the system.

4.6 Performance Scalability

Two important issues in the use of middleware are scalability and overheads. We conducted a set of experiments to determine the impact of executing the middleware on an off-the-shelve mobile device. We deployed the middleware onto a HP iPAQ h4150 PDA, which features Windows® Mobile™ 2003, Intel PXA255 400MHz, 64MB Ram, 32MB ROM and 1000mAh battery. We use Mysaifu JVM Version 0.3.3 [Mysaifu08] as the java runtime environment, and the middleware is deployed as a 1.1MB Java Archive (JAR).

4.6.1 QoS Parameter Scalability

As shown in Figure 4.11, we tested the platform using 8, 16, 32, 64 and 128 input QoS parameters. By hierarchically inferencing these QoS parameters, the setups contain 15, 31, 63, 127 and 255 primitive or abstract QoS factors correspondingly. The total numbers of fuzzy rules in these setups are 175, 375, 775, 1575 and 3175 correspondingly, which shows a linear increase as the number of input QoS parameters increases. The memory usage of the middleware shows an initial linear increase, while tapering as the number of QoS parameters go beyond 64. This is due to the effect of garbage collection of the JVM, which is able to free up memory of unused objects when the memory consumption is high. The corresponding memory usages are 11.16MB, 13.26MB, 18.11MB, 25.9MB and 31.99MB.

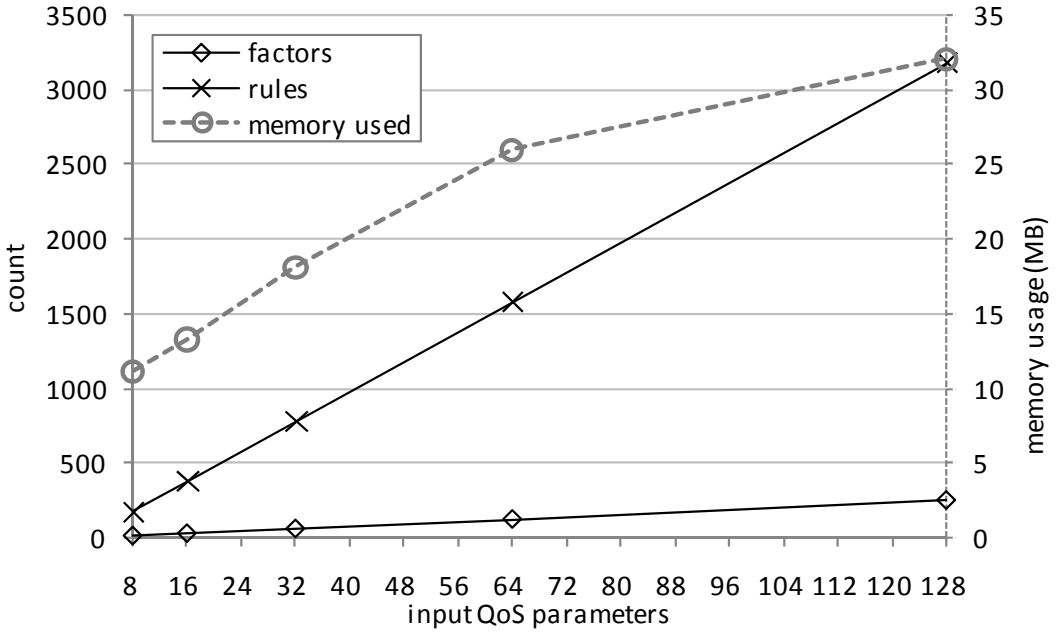

Figure 4.11 Parameter Size, Rule Size and Memory Usage

4.6.2 Fuzzy Rule Scalability

To further study the performance scalability of the middleware, each of the five setups is further divided into 3 cases. Corresponding to the 3 cases, the inputs are subjected to 50%, 25% and 12.5% of probability of change for each inference iteration. The values of changed inputs are randomized. Each of the 15 cases has been tested for extended times to achieve stabilized measurements in average affected rule counts and average execution times. As shown in Figure 4.12, both the execution time and the affected rule count are directly proportional to the number of changed input. We leverage the tree-structured QoS factor hierarchy by inferencing only the branch of rules whose inputs were changed since the last update. This technique significantly reduces the execution time when the operating environment is relatively stable.

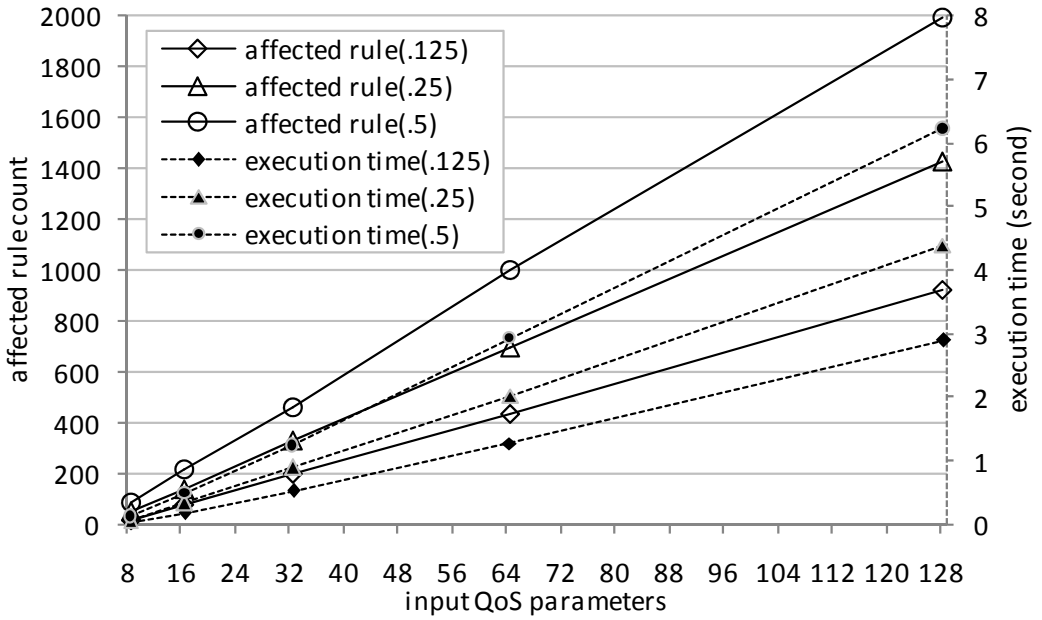

Figure 4.12 Performance Scalability of Fuzzy Rules

The execution times vary from 46ms for 22 affected rules to 6225ms for 1996 affected rules. This implies a 2 to 3 ms execution time per each affected rule. The execution time per rule is relatively shorter for smaller number of affected rules since the corresponding rule hierarchy is shallower and involves lesser number of intermediate abstract QoS factors. These QoS factors require extra de-fuzzification and re-fuzzification steps in-between different levels of rule hierarchies. The corresponding average execution time for 1996 rules is 3 ms on an AMD Althlon64 2GHz machine. This implies a 400 times of per CPU clock performance difference between the PDA and the AMD machine. This mainly due to the existence of the floating point processing unit of the AMD machine, which greatly accelerated the computation due to rules inference and de-fuzzification.

The battery life of the PDA is 4 hours and 7 minutes with maximum backlight under idle situation. By running the middleware with 32 inputs, 775 fuzzy rules, 12.5%

of input change probability and inferencing every 10 seconds, the battery life drops by 8.4% to 3 hours and 46 minutes.

4.6.3 Mobilet and Profile Scalability

The performance scalability of the number of mobilets and the number of mobilet service profile is independent of the number of QoS parameters and fuzzy rules. In terms of memory usage, each null mobilet consumes 15KB of memory, while adding more profiles to an existing mobilet only increase its memory usage marginally.

On the profile selection process, the computational overhead scales sub-linearly as the number of profile increases. The average profile testing and decision time for each profile is less than 1ms. Moreover, as a typical profile is only associated with a subset of the QoS factors, so that only a portion of all the profiles will go through profile testing to update their decision scores.

By running the middleware with 200 profiles, 32 inputs, 775 fuzzy rules and 12.5% of input change probability, the average execution time for profile testing and decision making is 83ms. Therefore, we consider the overhead caused by profile selection process insignificant.

Service reconfiguration can be one of the major overheads of the system. This process involves synchronizing the profile switching, insertion, and removal of mobilets. All of this requires the coordinated initialization, suspension, and termination of mobilet service objects at both ends of the wireless network

[Chuang05]. The latency of service reconfiguration caused by the middleware varies between 80ms and 1000ms. This latency means the application level data transmission is suspended for that period of time. E.g. an un-buffered audio clip streaming from the wired part to the mobile device will experience a 80ms to 1000ms interruption during service reconfiguration. However, if there is no application level data transmitting during the period of time, the application utilizing the middleware will experience no interruption. As mentioned in Section 4.4.2, we have limited the service reconfiguration to one mobilet at a time. This results in the service reconfiguration latency to be independent of the length of service chain and available mobilets. On the other hand, no measureable latency is found when the service reconfiguration only involves intra-mobilet profile switching.

4.7 Summary

This chapter describes a novel fuzzy knowledge-based QoS middleware framework for mobile and wireless environments. Special attention has been dedicated to the issue of how to deal with the problems of fuzzy rule explosion and multiple QoS objectives by employing the concepts of the fuzzy inference hierarchy and the multi-objective decision-making process. This chapter demonstrates the flexibility of our QoS management framework in adapting to different users, applications, and platforms operating in wireless environments that are characterized by dynamic and constrained resources. The proposed model provides an optimal overall service by synergistically balancing the QoS requirements of users and applications with the dynamic allocation of resources and chaining of services. Our research is novel in that it looks at QoS from a holistic,

systematic, and pragmatic perspective. In the next chapter, we will extend the framework to automatically and precisely map the preferences of the user and application developer onto the fuzzy control system, but without the need to rebuild membership functions or fuzzy rules.

Chapter 5

Meta-level Adaptation

The formulation of a fuzzy control model requires that the system designer understands how different input parameters affect the specific application. In the context of mobile applications, combing different users and applications bundles a unique set of requirements that are maintained and managed by the underlying system. However, these requirements often consist of runtime information that will not be available to the QoS management system until the application subscribes to the service of the QoS management system. Even worse, these requirements can change as users change their preferences during runtime. This is the reason why most existing fuzzy QoS control systems are application specific and lack the extensibility for adapting to different users, different applications, and different application usages. In contrast, MobiPADS is designed to adapt to not only the dynamism of the operating environment, but also to support the dynamic requirements of users, applications and application usages during runtime.

The ultimate goal for a mobile QoS management system is to maximize user-perceived QoS under a situation of limited computational resources. However, the perceptions of different users regarding different QoS factors are often subjective. For example, we can see in Figure 5.1 and Figure 5.2 two distinct visual effects of the same original frame, after it has been encoded correspondingly by MJPEG and LC codec mobilets and transferred through a network with a 5%

packet drop rate. In Figure 5.1, there are errors in the blue channel, which has caused three-fourths of the blue channel to be dropped and most of the frame to be dark and have a greenish tone. In Figure 5.2, there are errors in non-essential layers, so that some fine details of the frame have been lost and some error dots appear in the frame. Interestingly, under our simple fidelity measurement method, these two frames have the same fidelity score, but they appear dramatically different due to the behavior of the encoding scheme. Some users may prefer to have the fine details preserved without noise, while others may not want to have color shifting due to errors in a color channel. Therefore, to achieve a more accurate and flexible presentation-quality model, it would be desirable to allow users and application developers to control performance levels and to affect the relative weightings of the concerned QoS factors during runtime. This can be achieved by providing a set of QoS specification abstraction interfaces for users and applications while isolating the complexity of the fuzzy rule hierarchy from users and applications, such that the fuzzy rule bases can be easily and flexibly customized to match all of the detailed requirements of the user and applications. This translates to the need to support meta-level fuzzy model adaptation in MobiPADS during runtime. In the following sections, we describe the three meta-level adaptation mechanisms of MobiPADS, namely, *membership function adaptation*, *importance weight adaptation*, and *computational reflection*.

Figure 5.1 Visual Effect M-JPEG Enncoding with Packet Lost

Figure 5.2 Visual Effect of LC Encoding with Packet Lost

5.1 Membership Function Adaptation

We have proposed an adaptive fuzzy control architecture that flexibly supports new mobile applications. The architecture is comprised of a generic fuzzy QoS control model that predefines all the possible QoS parameters and defines the corresponding membership functions by heuristic approximation. All of the QoS parameters are normalized to match a scale composed of five linguistic values. The fuzzy rules are defined exhaustively to avoid the need for fuzzy rule interpolation. Although there are a large number of QoS parameters, the number of rules maintained by the system is limited because of the hierarchical organization of the QoS parameters.

By using this design, our fuzzy control model is able to support typical multimedia applications to react adaptively to the changes in the environment. However, the adaptation provided by the fuzzy control model at this stage is often sub-optimal since the membership functions of the QoS parameters are pre-modeled and different applications can have very different interpretations of the same QoS parameter.

In order to fully utilize the predefined fuzzy control model to support new adaptive applications, we introduce a number of operations that adapt the pre-defined membership functions to fulfill user and application requirements in a wireless environment. These operations are called *normal point shifting*. The idea behind normal point shifting is that, for each specific user and application combination, the system allows appropriate adjustments to the predefined membership functions to adapt to the corresponding interpretation on all the QoS

parameters, without the need to modify other parts of the generic fuzzy control model.

Normal point shifting allows the user and application developer to dynamically specify the normal position of a QoS parameter. As a result, the procedure eases and simplifies the adjustment of the membership functions that model the QoS parameters to more accurately reflect the interpretations of the user and application developer on that QoS parameter.

5.1.1 Normal Point Shifting

Three approaches for normal point shifting have been defined: *proportional shifting*, *reverse-proportional shifting*, and *relative-scaled shifting*. Figure 5.3 depicts the generic fuzzy sets of a QoS parameter before shifting, which typically represent five linguistic values – bad, poor, normal, good, and excellent.

Different types of QoS parameters require different shifting methods. Proportional shifting, which is shown in Figure 5.4, is suitable for QoS parameters that seek to bound the left-hand side of the fuzzy sets. Examples are frame rate and battery life, since these types of parameters are almost always interpreted as critical when their values approach 0%. On the other hand, an input value of 100% for frame rate or battery life may not necessarily be interpreted as excellent because users and applications may have different requirements. For instance, 100% for frame rate typically refers to 30 fps; however, when a user is viewing a high-speed sports video clip, 30 fps may only be considered good or even normal. Similarly, a user may plan to use a mobile device continuously for 3 hours, but the

device may have a maximum battery life of only 3.5 hours. In this case, a 100% battery life would only be considered to be normal by the user.

Figure 5.5 shows the opposite form of proportional shifting, called reverse-proportional shifting, which seeks to bound the right-hand side of the fuzzy sets. This type of shifting is suitable for QoS parameters that are always interpreted as excellent when the input approaches 100%, e.g., the connection cost of 3G wireless and inter-media synchronization. A 100% input value means a 3G wireless connection that is free of charge or perfectly synchronized media, which are always interpreted as excellent. On the other hand, when the input value approaches 0%, this can be interpreted as an extremely expensive 3G connection or media that are totally out of synchronization. In these cases, the interpretation of an input value approaching 0% is more abstract and dependent on the corresponding normalization functions and the requirements of users and applications than in the cases mentioned earlier.

Figure 5.6 shows the combined form of the previous two shifting types, which is called relative-scaled shifting. The interpretation of this type of QoS parameter is more objective in that both ends are bounded. Examples for such a parameter are CPU availability and network error condition. Since the raw measurements are either directly or reversely used as the input value, in the former case, 0% always means bad and 100% always means excellent for CPU availability; while in the latter case, a 0% error rate always means excellent and a 100% error rate represents the worst network condition, which means out-of-service.

Through the shifting of normal points of QoS parameters, users and application developers can dynamically and easily alter the pre-modeled fuzzy membership functions to better adapt to the requirements of users and applications.

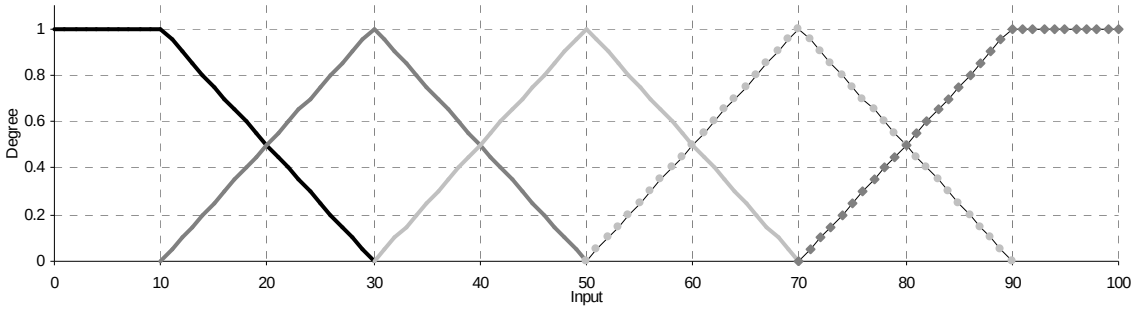

Figure 5.3 Generic Fuzzy Sets for A Qos Parameter

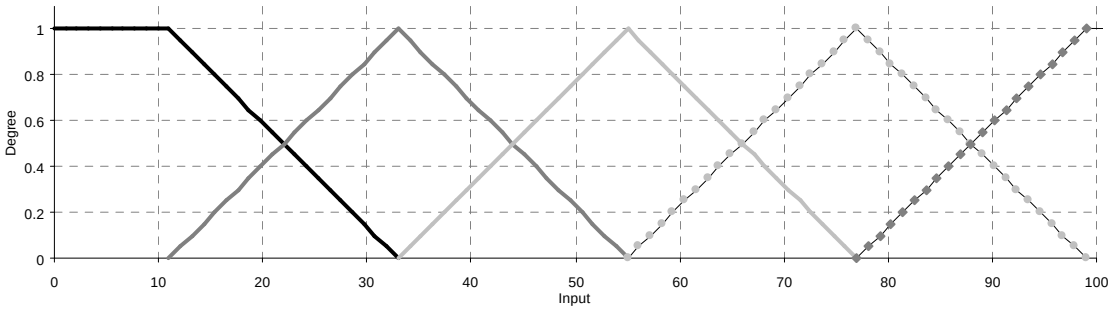

Figure 5.4 Proportional Shifting of Fuzzy Sets

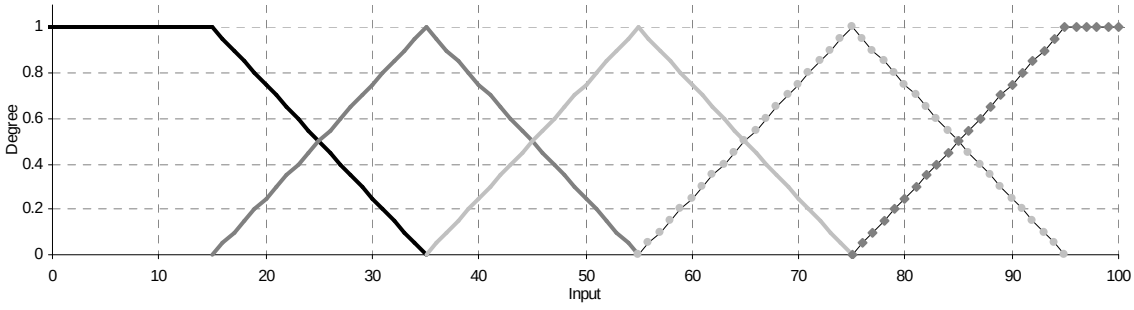

Figure 5.5 Reverse-proportional Shifting of Fuzzy Sets

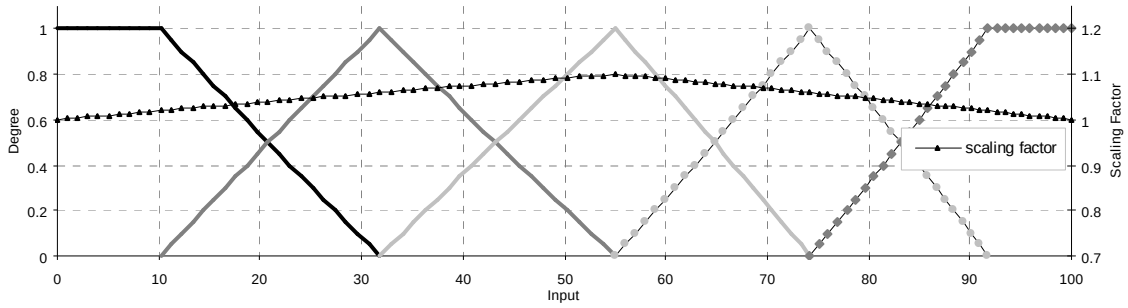

Figure 5.6 Relative-scaled Shifting of Fuzzy Sets

5.1.2 Experimental Results

Based on the experimental setup in Section 4.6, we conducted two sets of experiments to better understand the bahavior and impact of normal point shifting. The first experiment was configured with the normal point of the clarity parameter proportionally shifted to the right by 10%, while no normal point shifting was applied in the control experiment. The network bit error rate was set to 40% and the bandwidth was varied between 0 and 400 KB/s. The experiment exercised free adaptation on the mobilets of the service chain.

Figure 5.7 shows the results of the clarity performance. Clarity normal point shifting achieved an average of 3% better clarity than was found in the control experiment. The effect of clarity differences only appeared beyond 20% clarity, since the fuzzy sets of the clarity parameter were bounded at the left-hand side of the input value range. Figure 5.8 shows the smoothness performance of the two result sets. It is interesting to see that clarity normal point shifting did not significantly worsen the smoothness performance. On average, the clarity normal point shifting was only 0.16 fps worse than was seen in the control experiment. Figure 5.9 shows a similar finding that clarity normal point shifting

raised the packet drop rate by only 1.05% compared to what was seen in the control experiment.

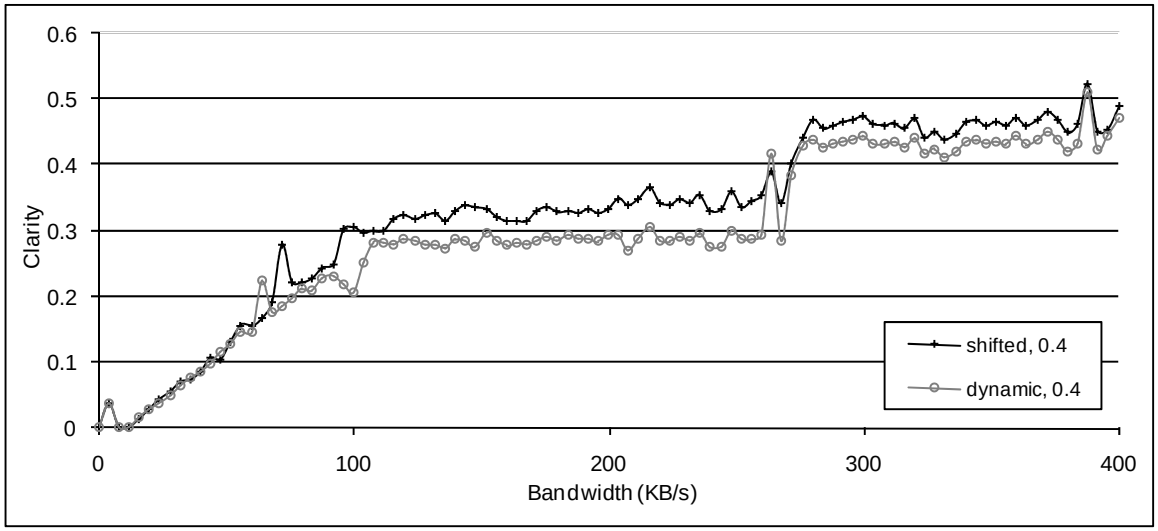

Figure 5.7 Clarity Performance with Clarity Normal Point Shifted

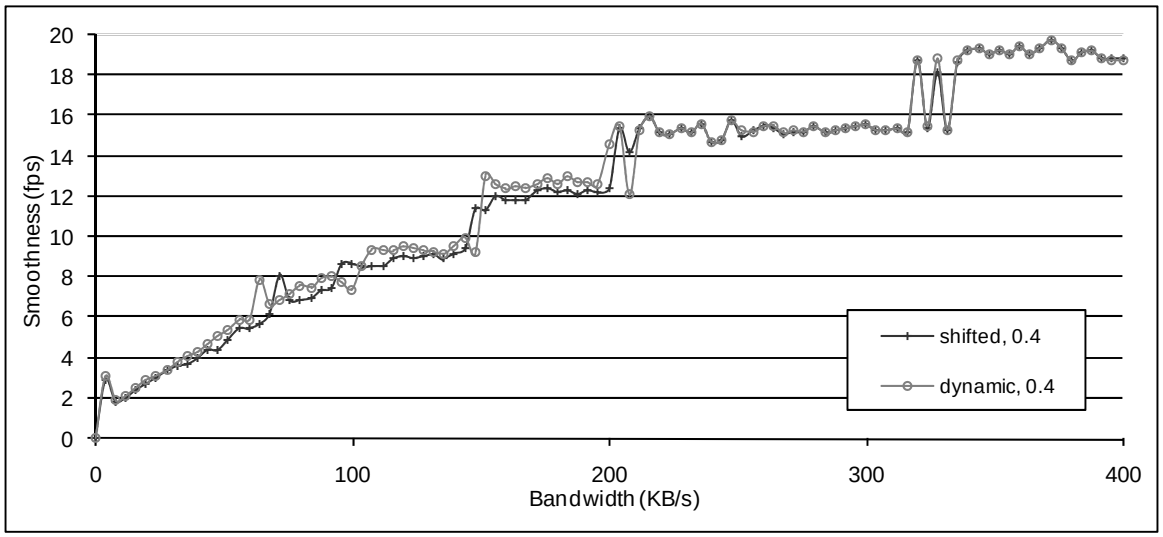

Figure 5.8 Smoothness Performance with Clarity Normal Point Shifted

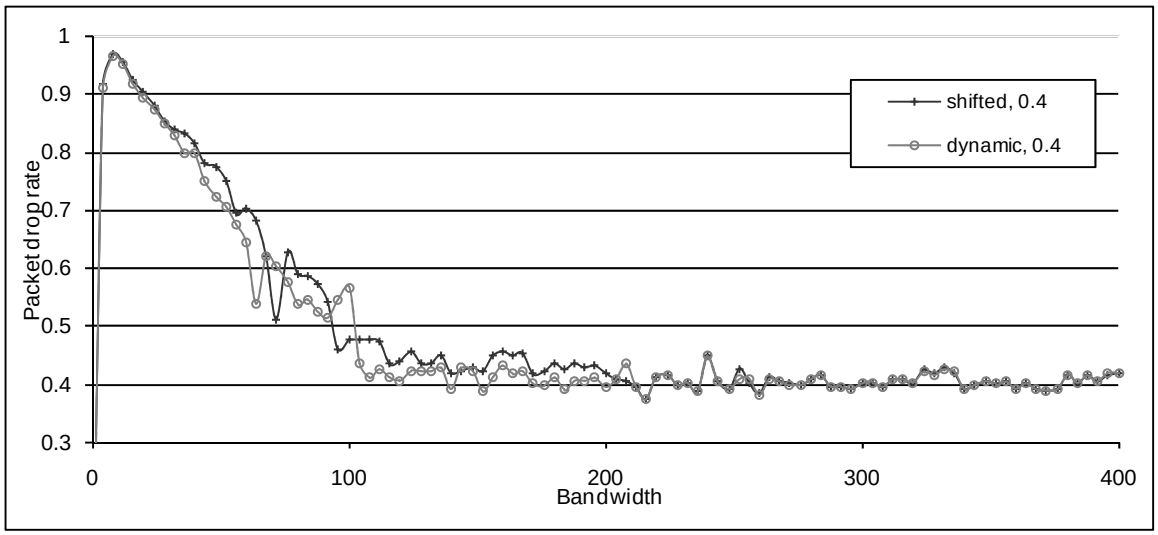

Figure 5.9 Packet Drop Rate with Clarity Normal Point Shifted

The second set of experiments was conducted with larger variations of normal point shifting but with a lowered network bit error rate of 10%. In this experiment, three result sets were collected. The first result set, labeled as *clarity shifted***,** was clarity driven with the clarity normal point shifted to the right by 10% and the smoothness normal point shifted to the left by 10%. The second result set, labeled as *smoothness shifted*, was smoothness driven and was collected by reversing the previous setting – the clarity point was shifted to the left by 10% and the smoothness point was shifted to the right by 10%. The third result set was the control experiment with no normal point shifting. The corresponding clarity, smoothness, and packet drop rate performances are shown in Figure 5.10, Figure 5.11and Figure 5.12.

Based on these two sets of experimental results, we found that normal point shifting does show effective inference on the concerned QoS aspects. However, it is worth noting that the normal point shift does not affect the QoS performance

directly, while shifting the membership function during fuzzification. This effect is cascaded through the various inference engines to the step for the selection of mobilet profiles, where a QoS aspect that does not attain the quality level specified in the normal point is given a higher weight. This leads the reconfiguration engine to choose mobile profiles in favor of improving the concerned QoS aspect. However, this does not guarantee the level of QoS improvement, nor will this effect be proportional across different resource levels. The reason for this is that, compared to the rate and level of changes in the resource and QoS level, the adaptation options and profile switching actions are relatively coarse-grained, so that the final adaptation effects will always fluctuate more than the changes in resource levels.

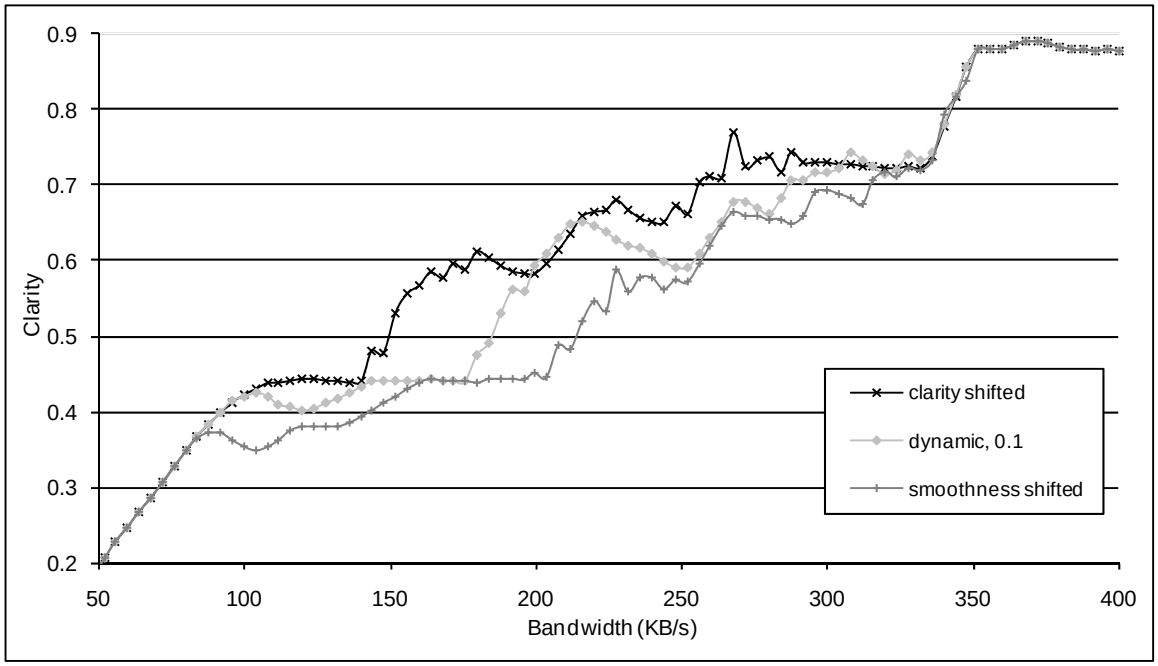

Figure 5.10 Clarity Performance of Clarity or Smoothness Normal Point Shifted

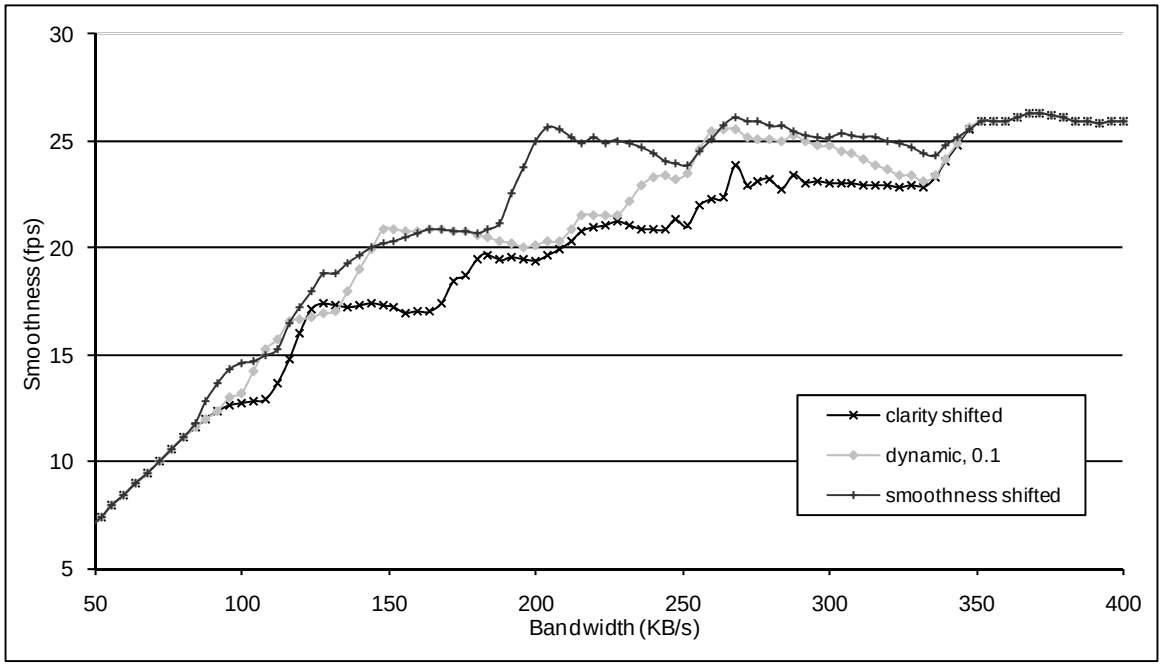

Figure 5.11 Smoothness Performance of Clarity Or Smoothness Normal Point Shifted

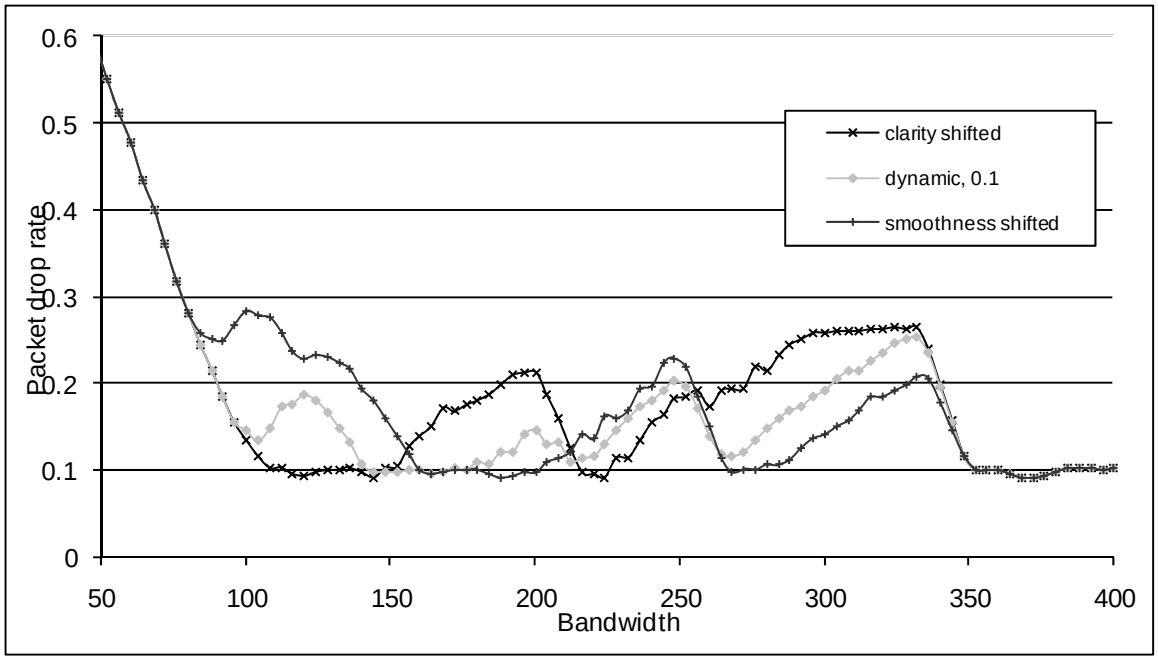

Figure 5.12 Packet Drop Rate of Clarity or Smoothness Normal Point Shifted

.

5.1.3 Discussion

Normal Point Shifting is a simple approach that can vastly improve the accuracy of the fuzzification process to closely model individual QoS factors based on the perceptions of the user and the demands of the application. However, even when all of the concerned QoS factors have been accurately modeled, users and application developers can have another dimension of demand for prioritizing the various concerned QoS factors, as seen in Section 5.1. In the following section, we will present another mechanism to fulfill this requirement for prioritizing QoS factors.

5.2 Importance Weight Adaptation

Attempting to impose and satisfy a hard-guarantee QoS contract operating in a wireless environment is difficult, to say the least. This is due to the extreme variations in the available resources and to the dynamic changing contexts of mobile devices and operating conditions. An alternative is to use priority-based approaches to differentiate between the different services in terms of importance, which is able to provide better QoS when compared to best effort services. The weighted priority approach introduces quantitative control of the CPU and network utilization of different services. Similarly, our model features Importance Weight Adaptation, which supports representing user and application requirements in the form of relative weightings assigned to different user satisfaction factors.

In Section 4.4, we described the use of relative importance values for the concerned QoS factors to support the selection of optimal mobilet profiles. This model assumes equal weighting among all of the concerned QoS factors, which may not

lead to the achievement of an optimal level of QoS as perceived by the user. This is because objective perspectives and different users are involved, as discussed in Section 5.1. The *adaptive fuzzy membership function* discussed in Section 5.2 partially addressed this issue from a microscopic approach that fine-tunes the normal point of fuzzy variables. In contrast, *importance weight adaptation* addresses the remainder of the issue from a more macroscopic approach.

5.2.1 Importance Weight for QoS Factors

Based on the preferences of users and the requirements of applications, users and application developers can assign a non-negative integral priority value to each of the concerned QoS factors. Initially, all of the concerned QoS factors have a priority value of 4, which is mapped to a weighting of 50%. Higher priorities will be mapped to weightings ranging from 60% to 100%, while lower priorities will be mapped to weightings ranging from 40% to 10%. By assigning different priorities to different concerned QoS factors, the importance values of each QoS factors are boosted or suppressed according to the requirements and preferences of users and applications.

This priority value **PV** ranges from 0 (lowest) to 9 (highest), and is denoted by

$$\mathbf{PV} = \{pv_1, pv_2, \ldots, pv_k\} \rightarrow [0, 9]. \quad (29)$$

The set of *importance weights* V representing the user-assigned and application-assigned QoS factor priorities, which is directly mapped from priority values, is denoted by:

$$\mathbf{V} = \{v_1, v_2, \ldots, v_k\} \rightarrow [0.1, 1], \text{ where } v_i = (pv_i+1)(0.1) \quad (30)$$

By augmenting the importance value *w* in (28), the *optimal profile* becomes

$$\delta_{\text{optimal}}(f^*) = \max_{f \in F(M)} \{ \min_{i=1}^{k} [\max(o_i(f), \overline{v_i w_i})] \}. \quad (31)$$

5.2.2 Experimental Results

Based on the experimental setup of 4.6, we conducted an experiment to study the bahavior and impact of the importance weight adaptation. The experiment was configured with importance values for the clarity parameter of 90%, 70%, 50%, 30%, 10%, while the importance value for other QoS factor was kept at 50%. The network bit error rate was 10% and the bandwidth varied between 0 and 400 KB/s. The experiment was configured to support adaptation of the mobilets composed along a service chain. The results of the clarity performance are shown in Figure 5.13.

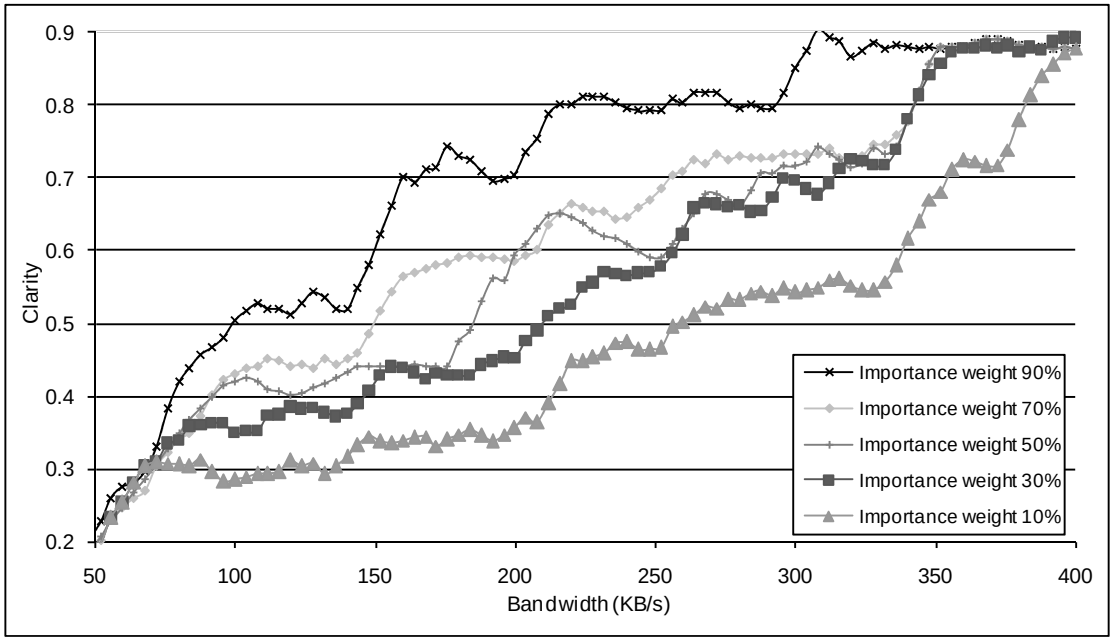

Figure 5.13 Performance of Clarity with Importance Weight Adaptations

As shown in Figure 5.13, the clarity performance varied proportionally to the changes in bandwidth under all importance weights. However, all of the five importance weight settings showed fluctuations in the trends lines. This was for the same reason as that discussed in Section 5.2.2, namely that compared to the rate and level of changes in the resource and QoS levels, the adaptation options and profile switching actions are relatively coarse-grained, so that the final adaptation effects will always fluctuate more than the changes in resource levels. The effect of a weight value higher than 50% results in a convex trend, while a weight value lower than 50% leads to a concave trend. From Figure 5.13, we can conclude that the larger the importance weight value of a QoS factor is, the better is the performance level of that QoS factor under a given resource level.

5.2.3 Discussion

Importance weight adaptation, coupled with *membership adaptation*, provides a concise set of adaptation mechanisms for users and application developers to customize the interpretations of QoS factors of different abstraction levels and from different perspectives. These two adaptation mechanisms virtually eliminate the need to predefine specific fuzzy models for different application scenarios, which generally requires expertise in fuzzy control and in the domain of applications. Specifically, these two mechanisms support the adaptation of the generic fuzzy model by referring to two sets of intuitive input parameters from the user and application: *preferred performance values* for concerned QoS factors, e.g., 15 fps in smoothness for video playback and 16-bit in fidelity for audio playback; and a list of *importance weight values* for these QoS factors, e.g., 90% in smoothness and 40% in fidelity. As a result, these mechanisms greatly reduce the difficulties in developing QoS-aware mobile applications that leverage the fuzzy control model.

5.3 Computational Reflection

While the adaptation mechanisms of MobiPADS offers a significant degree of freedom to customize the interpretation of QoS that guides the adaptation behaviors, at times mobile applications are still in the best position to make critical decisions on the operating context and hence the adaptation behavior. For this reason, MobiPADS provides the mobile application with an extensive set of APIs and reflective interfaces. Through meta-level object representations, a mobile application can gain access to the metadata of the internal fuzzy control model, service reconfiguration mechanism, and dynamic adaptation mechanisms of MobiPADS, and modify these entities to obtain optimal service provision from MobiPADS.

To present a clearer picture of the capabilities and roles of each entity in the MobiPADS platform in supporting adaptation, Table 2 shows the relationships between adaptation initiators and reconfigurable entities. There are two entities that can subscribe and react to changes in QoS factors: the *mobile application* and the *MobiPADS system* itself. On the other hand, there are three entities that can be reconfigured to adapt to the changes in the QoS factors: the mobile application, the service chain of MobiPADS, and the mobilet within the service chain.

A mobile application can respond to the contextual changes by changing its internal logic, or by changing the configuration of the service chain or even the behavior of individual mobilets within the service chain. By contrast, the MobiPADS cannot alter the internal logic of the mobile application, but may supply the mobile application with the necessary contextual information. In this

case, the application may choose to perform intra-application adaptation in reaction to the contextual information feed from the middleware.

Table 5.1 Relationships between Adaptation Initiators and Reconfiguration Entities

	Reconfigurable Entity		
Adaptation Initiator	Mobile Application	Service Chain	Mobilet
Mobile Application	✓ To adapt to the dynamic wireless environment, a mobile application can respond to the QoS factor and adjust its behavior accordingly.	✓ Using the application profile, a mobile application can specify the service configurations under different environments. The mobile application can also change the current configuration directly.	✓ By supplying suitable parameters to a mobilet, a mobile application can fine-tune the subtle behavior of individual mobilets, so that the most suitable mode of operation is selected.
MobiPADS System	The MobiPADS system cannot reconfigure the mobile application directly. It can only supply the mobile application with the context.	✓ According to the MobiPADS system profile, the configuration of the current service chain is actively adjusted to suit the existing environment.	✓ If specified in the system profile, the reconfiguration process can also switch the mode of operation of an individual mobilet by supplying suitable parameters.

5.3.1 Reflective API

To support the development of context-aware mobile applications, the MobiPADS exposes four meta-level objects that abstract the QoS interpretation, the service characteristics, the service reconfiguration, and the adaptation mechanisms of the system, which are shown in Table 3. Through these meta-objects, the mobile application can subscribe to the contextual changes, and is highly flexible in selecting and adjusting the service configuration and adaptation policy of the MobiPADS. The four meta-objects are the *ReConfigMeta*, *MobiletMeta*, *AdaptationMeta*, and *QoSMeta*. The respective roles of these meta-objects are listed below.

- The *ReConfigMeta* meta-object reflects the configuration of the current service chain that serves the mobile application. Through *ReConfigMeta*, a mobile application can subscribe to reconfiguration events and actively participate in the reconfiguration of the service chain.
- The *MobiletMeta* meta-object reflects the statuses and characteristics of

individual mobilet services. Through *MobiletMeta*, a mobile application can change the operating profiles of individual active mobilets.

- The *AdaptationMeta* meta-object reflects the input and control parameters of the fuzzy inferencing of MobiPADS. Through *AdaptationMeta*, a mobile application can modify the list of concerned QoS factors and also the fuzzy inference frequency among all inference engines. A mobile application can also subscribe to QoS factor events by specifying the notification conditions, e.g., by notifying the application when the QoS factor - battery level is less than 20%.
- The *QoSMeta* meta-object reflects the status and characteristics of individual QoS factors and, more importantly, it also reflects the key attributes of the two meta-adaptation mechanisms described in Sections 5.2 and 5.3 – *membership function adaptation* and *importance weight adaptation*. Through *QoSMeta*, a mobile application can reify the *membership function adaptation* mechanism by adjusting the value of the *normal point* and the approach chosen for *normal point shifting*. A mobile application can also reify the *importance weight adaptation* mechanism by modifying the *relative priority* value or the *absolute priority* value of the QoS factor. Moreover, a mobile application can also query the defuzzified value of the QoS factor.

Table 5.2 Reflective MobiPADS API for Context-Aware Mobile Applications

Interfaces	Description
interface **ReConfigMeta** {	***// Meta-object that reflects service reconfiguration***
String[] listAvailableServiceNames();	*// list the names of all available services*
String[] listActiveServiceNames();	*// list the names of services in the service chain*
MobiletMeta[] listAvailableService();	*// list all of the available services*
MobiletMeta[] listActiveService();	*// list the services in the service chain*
MobiletMeta getService(String mobiletName);	*// get a mobilet service object by name*
MobiletMeta getFirstService();	*// get the first mobilet service in the service chain*
MobiletMeta getLastService();	*// get the last mobilet service in the service chain*
void insertService(int position, MobiletMeta newService);	*// insert a mobilet service into the service chain*
void removeService(String serviceName);	*// remove an active mobilet service from the service chain*

String getCurrentServiceProfile(String mobiletName);	*// get the current service profile of an active mobilet service*
void setCurrentServiceProfile(String mobiletName, String	*// set the current service profile of an active mobilet service*
void reconfigure(MobiletMeta[] newServiceChain);	*// reconfigure the entire service chain*
int getMinimalTTL();	*// get the minimal time-to-live (ms) of a service chain*
void setMinimalTTL(int timeToLive);	*// set the minimal time-to-live (ms) of a service chain*
int getCurrentTTL();	*// get the current time-to-live (ms) of the service chain*
void setCurrentTTL(int timeToLive);	*// set the current time-to-live (ms) of the service chain*
void lockConfiguration();	*// disallow reconfiguration of the service chain*
void unlockConfiguration();	*// enable reconfiguration of the service chain*
void subscribeReconfiugration(ReconfigListener listener);	*// subscribe to the reconfiguration event*
void unscribe(ReconfigListener listener);	*// unsubscribe from the reconfiguration event*
}	
interface **MobiletMeta** {	***// Meta-object for mobilet***
String[] listServiceProfile();	*// list the service profiles available to the mobilet*
String getCurrentServiceProfile();	*// get the current service profile*
void setCurrentServiceProfile(String ServiceProfile);	*// set the current service profile*
String getName();	*// get the name of the mobilet service*
String getDescription();	*// get the description of the mobilet service*
boolean isActive();	*// check if the mobilet is operating in the service chain*
int getServicePosition();	*// get the position of the mobilet in the service chain*
void removeService();	*// remove the mobilet from the service chain*
String[] getInputTypes();	*// get the input media type of the mobilet*
String[] getOutputTypes();	*// set the input media type of the mobilet*
int getMinimalTTL();	*// get the minimal time-to-live (ms) of the mobilet*
void setgetMinimalTTL(int timeToLive);	*// set the minimal time-to-live (ms) of the mobilet*
int getCurrentTTL();	*// get the current time-to-live (ms) of the mobilet*
void setCurrentTTL(int timeToLive);	*// set the current time-to-live (ms) of the mobilet*
void lockService();	*// disallow reconfiguration of the mobilet*
void unlockService();	*// enable reconfiguration of the mobilet*
Map<String,Double[]> getAffectedFactors(String profile);	*// get the affected QoS Factors of a service profile*
void setAffectedFactors(String profile, Map<String,Double[]>);	*// set the affected QoS Factors of a service profile*
}	
interface **ReconfigListener** {	***// Listener for reconfiguration events***
public abstract void notifyReconfig(Date triggerTime, String[] oldChain, String[] oldProfiles, String[] newChain, String[] newProfiles);	*// invoked before reconfiguration. Inform the application about the changes to be made in the service chain*
}	
interface **AdaptationMeta** {	***// Meta-object that reflect the adaptation mechanism***
QoSMeta[] listAllQoS();	*// list all QoS factors*
QoSMeta[] listConcernedQoS();	*// list the QoS factors concerned by the current application*
String[] listAllQoSNames();	*// list the names all QoS factors*
String[] listConcernedQoSNames();	*// list the names of the concerned QoS factors*
QoSMeta getQoS(String QoSName);	*// get a QoS factor meta-object by name*
QoSMeta getContextRootQoS();	*// get the root node for contextual QoS factors*
QoSMeta getUserRootQoS();	*// get the root node for user QoS factors*
int getInferInterval();	*// get the inference interval of the whole fuzzy model*
void setInferInterval(int interval);	*// set the inference interval of the whole fuzzy model*
void subscribeQoS(QoSListener listener, String qosName, String relation, double refValue);	*// subscribe to a QoS event, with condition of notification*
void unsubscribeQoS(QoSListener listener);	*// unsubscribe a QoS event*
}	
interface **QoSMeta** {	***// Meta-object for QoS factor***
QoSMeta[] getQoSComposition();	*// get the child QoS factors that this QoS factor is referring to*
boolean isLeaf();	*// check if this is a leaf node*
boolean isContextRoot();	*// check if this is the root node of contextual QoS factors*
boolean isUsertRoot();	*// check if this is the root node of user QoS factors*
boolean isConcerned();	*// check if this is a QoS factor concerned by user or application*
void setConcerned(boolean concernFlag);	*// set this as a QoS factor concerned by user or application*
String getName();	*// get the name of the QoS factor*
String getDescription();	*// get the description of the QoS factor*
double getValue();	*// get the defuzzified QoS value of the QoS factor*

int getUpdateInterval();	*// get the inference and update interval of the QoS factor*
void setUpdateInterval(int interval);	*// set the inference and update interval of the QoS factor*
double getNormalPoint();	*// get the normal point of the QoS factor*
void setNormalPoint(double value);	*// set the normal point of the QoS factor*
String getNPShiftingType();	*// get the type of normal point shifting approach*
void setNPShiftingType(String shiftingType);	*// set the type of normal point shifting approach*
int getPriority();	*// get the priority value of the QoS factor*
void setPriority(int priority);	*// set the priority value of the QoS factor*
double getImportanceWeight();	*// get the importance weight of the QoS factor*
void setImportanceWeight(double weight);	*// set the importance weight of the QoS factor*
int getRelativePriority();	*// get the relative priority rank of the QoS factor among all concerned QoS factors*
void setRelativePriority(int priority);	*// set the relative priority rank of the QoS factor among all concerned QoS factors; the priority values of all concerned QoS factors will be automatically adjusted*
}	
interface **QoSListener** {	***// Listener for QoS events***
public abstract void notifyQoS(Date triggerTime, String detail);	*// invoked when the predefined monitoring condition on the QoS factor is fulfilled. Inform the application about the trigger*
}	

5.3.2 A Case Example

To present a clearer understanding of the meta-objects, we give an example in Table 5.3 of how a mobile Web application can 1) adjust the adaptation mechanisms of MobiPADS and 2) adjust its internal logic and the service chain of MobiPADS, in response to environmental changes. Using the MobiPADS reflective API, the sample application specifies two concerned QoS factors – *power availability* and *media fidelity*, and configure their normal points and priorities. As such, the adaptation mechanisms of MobiPADS will emphasize these two QoS factors, while aiming to attain the specified normal point values for these two QoS factors. The application also subscribes to a *battery level* QoS event, through which MobiPADS will notify the application when the battery level is below 20%. Upon being notified, and if the mobilet is deployed in the current service chain, the application will enforce the removal of the multiple description coding mobilet to reduce CPU loading, which correspondingly reduces power consumption.

Table 5.3 shows the sample code of the mobile Web application. On lines 2-13,

the *setAdaptation()* method shows the way to specify concerned QoS factors to the MobiPADS. Lines 4-7 add *PowerAvailability* as a concerned QoS factor and specify the normal point and the priority of this QoS factor. Similarly, lines 8-11 specify *MediaPriority* as another concerned QoS factor. Line 12 subscribes to a QoS factor – *BatteryLevel* – and specifies the notification condition of the battery level if it falls below 20%.

Table 5.3 Sample Context-aware Mobile Application

```
 1 public class SampleApp {
 2   public void setAdaptation() {
 3     AdaptationMeta adapt = MobiPADS.getAdaptationMeta();
 4     QoSMeta power = adapt.getQoS("PowerAvailability");
 5     power.setConcerned(true);
 6     power.setNormalPoint(0.5);
 7     power.setPriority(1);
 8     QoSMeta fidelity = adapt.getQoS("MediaFidelity");
 9     fidelity.setConcerned(true);
10     fidelity.setNormalPoint(0.7);
11     power.setPriority(2);
12     adapt.subscribeQoS(MyListener, "BatteryLevel", "LESS_THAN", 0.2);
13   }
14
15   class MyListener implements QoSListener {
16     public void notifyQoS(Date triggerTime, String detail) {
17       ReConfigMeta cfg = MobiPADS.getReConfigMeta();
18       MobiletMeta mdc = cfg.getService("MultipleDescriptionCoding");
19       if (mdc.isActive) {
20         mdc.removeService();
21         mdc.lockService();
22       }
23     }
24   }
25 }
```

Lines 15-24 defines the *Listener* class for the subscribed QoS event. On lines 16-23, the implementation of *notifyQoS( )* specifies that once the condition for this listener is fulfilled and the event is triggered, it will look up the mobilet *MultipleDescriptionCoding* in the *ReConfigMeta* meta-object. If the mobilet exist in the current service chain, it will be removed and prohibited from re-entering the service chain again during the current application section.

5.4 Summary

The fuzzy control model of a fuzzy control system is typically modeled by domain experts. Extending our previous works, our proposed fuzzy QoS control model features *membership function adaptation* and *importance weight adaptation*, which are designed to reduce the effort of mobile application developers. Otherwise, mobile application developers would have to put much more effort into understanding and controlling every detail of the contextual environment. By using *normal point shifting* and *importance weight*, a mobile application developer only needs to be aware of relevant contextual details down to a level that is sufficient for making adaptation decisions. This alleviates the need for developers to manage low-level contextual parameters. The experimental results also showed that membership function adaptation and importance weight adaptation can effectively tune the performance of individual QoS parameters under a resource-limited environment to adapt to the needs of different users and applications. However, although these two adaptation mechanisms offer a significant degree of freedom with regard to QoS adaptations, mobile applications are sometimes still in the best position to make critical decisions on operating context and hence adaptation mechanisms. For this reason, MobiPADS provides mobile applications with an extensive set of APIs and reflective interfaces. Through the meta-level object representation of the internal fuzzy control model and service reconfiguration mechanism, a mobile application can access the QoS information, service configuration, and adaptation mechanisms of MobiPADS, and modify these entities to obtain optimal service provision from MobiPADS.

In this chapter, we have presented three meta-level adaptation mechanisms that offer a principled approach to helping users and application developers to easily specify and control desired adaptation policies from different perspectives. They are: *Membership Function Adaptation*, *Importance Weight Adaptation*, and *Computational Reflection*. Importantly, by coupling with each other, these three mechanisms are able to accurately, efficiently, flexibly, and holistically map user and application preferences and requirements into the adaptation behaviors of the QoS control model.

Chapter 6

Related Works

This section presents a number of frameworks and architectures that provide QoS management support for applications with QoS requirements. We summarize their features and analyze their suitability for deployment over a mobile operating environment.

6.1 OMEGA

The OMEGA architecture [Nahrstedt96] is a QoS architecture that provides real-time guarantees in distributed multimedia systems. The research effort has been focused on resource management from both local and global perspectives.

The QoS Broker [Nahrstedt95] is the core component of OMEGA. It is a middleware responsible for the negotiation of QoS levels to be delivered to the application by the underlying system. The QoS Broker translates the requirements specified by the application, which then negotiates the resource allocations with the operating system and the network.

QoS Broker employs a set of translation relations for each media type, translates the application-level parameters specified by the user into lower-level QoS requirements. Subsequently, the QoS Broker follows these low level requirements and reserves resource based on QoS parameters associated with network and operation system resources, both local and remote. The translation is bi-directional, such that changes in resource reservations can be dynamically

reported to the user as application-level QoS parameters. A local QoS Broker aiming to perform reservations of remote resources, called a *buyer*, is responsible for interacting with other remote QoS Brokers, known as *sellers*. The architecture employs a *buyer/seller protocol* to allow sellers to advertise their services and buyers to contact sellers and reserve resources of sellers.

The QoS Broker provides orchestration service for balancing resource usage. The orchestration service utilizes information stored in resource databases to adjust the balances among resources of multimedia devices, operating systems, and the network. The architecture assumes the underlying operating system to have real-time capabilities, which allows the QoS Broker to predict and leverage the temporal behaviors of the OS for performing the orchestration of resources.

The OMEGA Architecture adopts a communication model that consists of two protocols at application and network levels. The Real-Time Application Protocol (RTAP) implements functions for call management, device management, synchronization, and media delivery at the application level. The Real-Time Network Protocol (RTNP) is responsible for connection management, error correction, rate control, and network access at the transport level. By using these two protocols, the OMEGA architecture provides guaranteed communication services over specified communication channels to applications.

The OMEGA architecture supports resources reservations at the operating system and network levels. It also provides a complete QoS translation mechanism that makes the underlying low level resources transparent to the application.

6.2 QoS-A

QoS-A [Campbell96], developed by the Distributed Multimedia Research Group at Lancaster University, is a QoS Architecture for specifying and implementing performance properties of multimedia applications over ATM-based networks. The architecture provides QoS mechanisms that span across all architectural layers, including end-systems, communications systems, and networks.

The architecture incorporates the notions of *flows*, *service contracts* and *flow management* to an ATM networked environment. Flows characterize the production, transmission and eventual consumption of single media streams, service contracts are binding agreements between users and providers and flow management provides for the monitoring and maintenance of the contracted QoS levels.

QoS-A aims to provide data flows with an associated level of QoS through tight coupling of devices, end-systems, and networks. QoS-A provides a QoS-configurable communication mechanism by using an augmented layers and planes structure integrating the existing layers and planes of the ATM architecture. It also employs thread-scheduling algorithms based on QoS constraints to achieve desired behaviors at system level. Devices are also built with QoS capabilities to support both scheduling and communication mechanisms.

The QoS-A architecture is composed by three planes, *protocol*, *QoS maintenance* and *flow management*. The protocol plane is responsible for data transfer. It consists of a user plane for transmitting media data and a control plane for carrying

control data. The QoS maintenance plane is responsible for monitoring and maintaining the QoS levels specified in the service contract that has been established among the user and the architecture. The flow management plane is responsible for admission control resource reservation, flow establishment, QoS renegotiation, QoS mapping and translation, and QoS adaptation.

The protocol and QoS maintenance planes are subdivided into four layers, *distributed systems platform, orchestration layer, transport layer* and *lower layer*. The distributed systems platform is responsible for providing services for QoS specification and multimedia communication. The orchestration layer supports media synchronization and jitter correction. The transport layer provides QoS-configurable communication service. The lower layers, which include network, data link and physical layer, provide low level communication services.

QoS-A is implemented on top of the Chorus open microkernel architecture, which focused on deploying multimedia protocols for local ATM network. Similar to OMEGA, QoS-A also supports translation between different levels of QoS requirements and resource reservations. However, QoS-A supports transparent QoS adaptation dynamically while OMEGA can only notify the application when QoS changes.

6.3 QuO

The Quality of Service for CORBA Objects (QuO) architecture [Quo08] provides QoS abstractions that can be utilized by distributed CORBA objects. QuO supports a QoS Description Language (QDL) by extending the Interface

Description Language (IDL) of CORBA.

The QDL language encapsulates QoS representations into object abstractions. QDL supports the description of resource requirements of CORBA objects. The language can also express the resources available in the system and their statuses. QDL also supports QoS contracts between a client and a server object, which includes the requested level of service, the level of service that is achievable, and actions to be taken when the QoS level changes. QDL supports the notions of QoS regions that allow the application to adapt to changing network conditions by changing from one QoS region to another. The architectural components of QDL are regular CORBA objects that are synthesized using IDL and QDL descriptions. These architectural components are responsible for QoS enforcement, QoS measurement, and QoS adaptation.

Although the design of a QoS specification framework should logically separate from QoS provision, the lack of knowledge of available QoS mechanisms hinders the practical feasibility of existing QoS specification languages. QuO integrates AQuA [Kuhns99] for dependability and TAO [Cukier98] for real-time support allowing for the provision of various QoS properties. This provides a concrete link between QoS requirements and QoS provision, pragmatically strengthening the usability of QuO.

6.4 HQML

The Hierarchical QoS Markup Language (HQML) [Gu01], developed by the MONET research group at the University of Illinois at Urbana-Champaign, is an

XML-based QoS language for supporting distributed multimedia applications. Hierarchical refers to the three levels of specification in HQML, which are the user level, application level, and system resource level. The MONET group has also conducted an extensive survey [Jin04] of existing QoS specification languages.

HQML supports adaptive middleware for improving QoS provision through application layer tags including *ReconfigRule*, *Condition*, and *Action*. When adaptation occurs, HQML allows feedback to and from the user through the *Notification* and *Feedback* tags.

The system resource level in HQML includes parameters from network, CPU, memory, and disk. However, application developers are not required to directly deal with these parameters. Instead, HQML provides a visual programming environment for developers to generate HQML files and a compiler that maps application level specifications resource level. A boundary symbol relational grammar named ConfigG, is used to check for formal consistency on visual QoS specification and automatic HQML generation. Unlike most of the QoS middleware, which focuses on communication models and low-level resource management, the strength of HQML is the inclusion of user-level QoS specifications.

6.5 OWL-S

OWL-S [DAML08], is an OWL ontology aims at incorporating Semantic Web techniques for unambiguously describing Web Service semantics in machine

interpretable representations. These descriptions can be used for dynamic service composition based on formally specified service QoS specification.

In OWL, resources with similar characteristics were grouped using the abstraction of *classes*. The OWL-S ontology is structured on top of the Service class. A Service class contains three properties, which are *ServiceProfile*, *ServiceModel* and *ServiceGrounding*. *ServiceProfile* describes what resources the service provides and requires. *ServiceModel* describes how the service works. *ServiceGrounding* describes how to use the service. The *ServiceProfile* class is composed of contact information and functional descriptions including preconditions, inputs, outputs and effects. Each *ServiceProfile* class is also associated with feature descriptions including service category, quality rating and parameter list. However, OWL-S does not provide nor adopt any specific scheme for both service category and quality rating. Therefore, it is up to the provider and user to ensure the adoption of the same categorization and rating scheme. Parameter list is unbound and can carry any information. This provides the flexibility for extending the *ServiceProfile* class in an unstructured manner.

DAML-QoS [Zhou04] leverages the OWL-S Service class to specifically provide QoS representation. This is done by extending the *ServiceProfile* class with *QoSProfile* class to provide a QoS ontology complementing OWL-S. QoS metrics and property constraints have been added to support the matching between service and service user. DAML-QoS has also introduced a matchmaking algorithm for supporting different matching degrees based on the constraints. However, the multidimensional QoS constraints can hardly be

utilized in a dynamic mobile environment, since the constraints can be easily violated under the fluctuating QoS resource availabilities. The effect of fluctuating QoS resource levels is thus amplified by the temporary unavailability of services due to unmatched constraints. This can result in a user perceived QoS that is significantly poorer than unmanaged best effort services.

6.6 QML

The QoS Modeling Language (QML) [Frølund98], developed by Hewlett Packard, is a general-purpose language for describing the QoS properties of software components. The research focus of QML is on QoS specifications and does not deal with other aspects of QoS management.

QML is designed to specify multiple QoS aspects such as reliability, performance, security, and timing. QML uses the abstractions of *contract type*, *contract* and *profile*. Contract type defines the properties associated with a specific QoS aspect. For example, the performance contract type is defined by the delay and throughput properties. Each property is bounded to a domain of values that may be specified numerically, as a set or as an enumeration. The favorable ordering of a property value, whether greater is better or smaller is better, is specified by the increasing and decreasing keyword. A contract is an instance of contract type, specifying the constraints on all the properties. A profile is a composition of contracts for a service, specifying the requirements on properties of all QoS aspects for utilizing the service.

QML supports refinements of contacts and profiles. A refinement is similar to

inheritance in object-oriented programming, in which a child refines a parent by inheriting the property constraints of the parent. However, a child contact or profile can modify the constraints of its parent by further tighten these constraints, or adding more constraints, but not loosen them.

QML lacks the ability for specifying the relationships, dependencies or tradeoffs among the properties of a contact. This greatly limits the flexibility for the language to support dynamic adaptation. When some of the requirements cannot be fulfilled during runtime, it would be useful to have the interdependency information for supporting decision making on adapting the services to the variations in QoS resource levels.

6.7 SLAng

SLAng [Lamanna03] is an XML-based language for describing service level agreements (SLA). SLAng focuses on providing a language model for supporting QoS negotiation and contract specification.

SLAng provides different levels of QoS service level abstraction, including application level, middleware level and the underlying resource level. SLAng also introduced another dimension of expressiveness called for *Horizontal* and *Vertical SLAs*. Horizontal SLAs are contracted between different parties providing the same kind of service. Horizontal SLAs govern the interaction between these coordinated peers. For example, two container providers can collaborate for replicating components. On the other hand, vertical SLAs are contracted between subordinated pairs. Vertical SLAs regulate the support parties get from their

underlying infrastructure. For example, a container provider can specify an agreement with an ISP for network services. The structure of an SLA in SLAng begins with an indication of horizontal or vertical agreement type. The SLA then specifies responsibilities in three sections: client, server, and mutual. In each of these sections, parameters specific to the SLA type are defined and therefore limited to those defined in the language itself.

SLAng not only supports QoS specification on application, middleware and resource levels, it also provides SLA for service peers as well as subordinated pairs. This makes SLAng a suitable input for QoS-aware adaptive middleware and automated reasoning systems.

6.8 Comparisons

Table 3.1 summarized the features of the reviewed QoS specifications and compared them against MobiPADS. Note that what is listed in the table is only an approximation; different projects have different focuses and are presented differently and it is difficult to measure the degree of completeness of a feature provide by a framework.

Table 6.1 Comparisons among Different QoS Frameworks

	User Specification	Application Specification	Resource Specification	Dynamic Adaptation	Negotiation	Specification Translation
OMEGA	No	Yes	Yes	Yes	Yes	Yes
QoS-A	No	Yes	No	Yes	No	Yes
QuO	No	Yes	Yes	Yes	No	No
HQML	Yes	Yes	Yes	Yes	No	Yes
OWL-S	No	Yes	Yes	No	Yes	No

QML	No	Yes	Yes	No	No	No
SLAng	No	Yes	Yes	No	Yes	No
MobiPADS	Yes	Yes	Yes	Yes	No	Yes

Among all of the frameworks reviewed, MobiPADS supports all dimensions of specifications and is able to adapt to these specifications dynamically and effectively. Even though MobiPADS does not support explicit *QoS negotiation*, this is not considered a major drawback. This is because in a mobile operating environment with dynamic and unpredictable resource levels, the underlying QoS framework is simply unable to guarantee the resource allocation and thus defeats the objective of the QoS negotiation to reserve resources for a specific application session.

6.9 Other QoS Middleware

XQoS [Exposito02] is an XML-based language for QoS specification that applies mainly to multimedia systems, one of its main observations being the need for both intra and inter-flow QoS specifications. The basis of the language is the Time Stream Petri Network (TSPN) model. This model is particularly suitable for modeling synchronization issues in concurrent streams or processes. The concentration on stream-based QoS limits its flexibility for operating in a mobile environment.

The Quality Assurance Language (QuAL) in QoSME [Florissi96] is similar to XQoS in that it is based upon the Time Stream Petri Network (TSPN) formal model. It facilitates optimal QoS mapping from application level requirements to underlying communications service specifications.

Service Level Specifications (SLS) [TEQUILA08] are being standardized as part of the TEQUILA project. In this context an SLS refers to network QoS in a public IP network and applies to a uni-directional traffic flow.

The need to create a more robust and configurable middleware system is realized in the development of the Open ORB [Blair01]. The design and implementation of Open ORB is based on a reflective middleware platform. Access to the underlying platform is achieved through the meta-interface, which exposes the meta-space that represents the support environment for the component. The Open ORB architecture is designed as a general middleware that supports system re-configurations by allowing applications to inspect (through reflection) and to adapt (through reification) the behavior of the underlying system components. The architecture does not have specific mechanisms to support mobile computing applications so that context awareness can be seamlessly integrated into the programming model to facilitate the dynamic configuration and deployment of mobile services.

CARISMA [Capra03] makes use of reflection to support the interactions between a mobile application and the middleware. Each application has a profile installed in the middleware, which contains policies that specify how contextual changes should be handled by the middleware. In case of conflict, CARISMA leverages a microeconomic approach that performs a "closed-bid" action to decide its adaptation action. CARISMA requires an application code to dynamically update its corresponding policies through reflection to manage QoS.

Work carried out at Illinois has led to the development of the Universal Interoperable Core (UIC) [Roman01], which is a reflective middleware platform designed for handheld devices. The platform can adopt different middleware personalities, e.g., a SOAP server and a CORBA server.

The Reflective Middleware for Mobile Computing (ReMMoC) [Grace03] platform demonstrates a similar approach that adapts an asynchronous middleware design and heterogeneous discovery protocols.

The Mobiware toolkit [Angin98] is built on CORBA and Java distributed object technology. Mobiware can run on mobile devices, wireless access points, and mobile-capable switches and routers. Mobiware provides API and algorithms for adaptive mobile network services, including QoS controlled handoffs, soft-state mobile QoS reservations, and flow bundling.

In the Rover toolkit [Kaashoek97], the middleware supports the development of both mobile-transparent and mobile-aware applications. Based on client-server architecture, the Rover toolkit provides a distributed object system for the development of mobile or distributed applications. Rover client applications typically run on mobile hosts, but can also run on fixed hosts. Server applications run on fixed hosts and maintain the long-term state of the system. The key idea of Rover is the introduction of the relocatable object (RDO) and the queued remote procedure call (QRPC). Object codes extending from RDO can be easily relocated from the server to the mobile client (or vice-versa) to allow for disconnected

operations. On the other hand, QRPC permits applications to perform remote procedural calls even when a connection is unavailable, by queuing the calls locally and serving as the connection is reestablished.

CARMEN [Bellavista03] supports context-dependent services for the wireless Internet. CARMEN uses profiles to describe the characteristics of any resource modeled in the system, including the user, service, device, and operating platform. By integrating different types of high-level metadata, CARMEN hides low-level mechanisms and implementation details from service developers and system administrators, while providing management configurability.

The EgoSpaces [Julien06] middleware provides information on context to applications in an abstract form. It adapts an agent-based approach that allows agents to define their own operating context and adaptation actions in response to a change in content.

The EasyLiving project [Brumitt00] focuses on the development of intelligent environments. The project identifies several research aspects, including middleware, geometric world modeling, sensing capabilities, and service description.

The Gaia project [Roman02] is a distributed middleware infrastructure that provides support for ubiquitous computing. The main intended application domain of Gaia is restricted to fixed intelligent environments and lacks the support for nomadic scenarios.

6.10 Summary

Mobile middleware show encouraging results in both performance improvements and value-added services for mobile applications. However, the lack of flexibility in the QoS models of current middleware systems limits their adaptability when facing dynamic requirements from user and mobile applications. In contrast, MobiPADS looks at QoS from a holistic, systematic, and pragmatic perspective. This thesis demonstrates the flexibility and efficiency of our QoS management framework in adapting not only to a dynamic operating environment and constrained resources, but also to the dynamic QoS requirements of user and application during runtime.

Chapter 7

Conclusions and Future Work

Existing Internet protocols and traditional reservation-based QoS management mechanisms were designed under the assumptions of high bandwidth and stable connectivity, provided when using a fixed network environment. These assumptions are invalidated when users move to mobile connectivity, where the operating environment is far more hostile. Mobile QoS management is hindered by the problems of highly variable connection quality, the high cost of connections, limited computational power, and a short battery life on portable systems.

The major task of mobile QoS management is thus to balance and make tradeoffs between various QoS parameters. However, mobile operating environments involve a large number of QoS parameters, and the characteristics of different system resources and user-perceivable aspects can be very diverse. This makes it difficult to deduce cost functions and to formulate an analytical optimization model for mobile QoS management. On the other hand, the combination of different users and different applications form a specific unique set of requirements. However, these requirements are often runtime information that will not be available to the QoS management system until the application subscribes to the service of the QoS management system. Moreover, these requirements can be subject to change as users change their preferences during runtime.

In order to answer the call for multi-dimensional adaptation needs raised by the diversity and dynamics of operating environments, mobile devices, users,

applications, and usage scenarios, we introduced the MobiPADS system. In this thesis, we described the design and implementation of the hierarchical fuzzy control model, upon which the adaptation mechanisms of MobiPADS were built. The MobiPADS system abstracted and organized contextual-based and user-perception-based QoS parameters within the hierarchy of a fuzzy model. This allowed all QoS inputs to be inferred and contribute to the adaptation decisions. Moreover, the hierarchical organization of fuzzy inputs not only avoided the *rule-explosion* problem, but also allowed new QoS inputs to be efficiently added to the hierarchy, without the need to review the entire fuzzy rule base in the process. The MobiPADS system introduced the following five adaptation mechanisms for flexibly supporting adaptation needs at different service levels.

- **Fuzzy Based Mobile Service Reconfiguration** adapts to the system dynamics and optimizes the QoS perceived by the user by reconfiguring a chain of mobilet services. Based on the inferencing of the fuzzy QoS factor hierarchy, this mechanism responds to the system dynamics in real-time by periodically selecting the optimal set of mobilet service profiles. Mobilet services are chained to provide an assortment of services to a mobile application. To adapt to vigorous changes in the QoS requirements, MobiPADS supports intra-mobilet reconfiguration that changes the mode of operation internally, as well as inter-mobilet reconfiguration that adds or removes mobilets on demand.

- **QoS Factor Hierarchy Extension** allows new QoS dimensions to be added to the existing QoS factor hierarchy, such that the number and variety of

managed QoS parameters will not be bound by the original design. Traditionally, adding a new fuzzy input to an existing fuzzy model meant that the whole model needed to be revamped. In contrast, under this extension mechanism, adding a new QoS factor will only cause local changes to be made to the overall QoS factor hierarchy. Importantly, this mechanism only causes a minimum amount of disruption to the entire fuzzy rule base, and updates only the direct parent node of the new QoS factor within the hierarchy. This makes unnecessary much of the effort required for a non-hierarchical fuzzy rule base to review the whole rule base when new input is added.

- **Membership Function Adaptation** allows ad-hoc adjustment on the interpretation of QoS factors to align with the specifications of a new application. This is done by using a single normal point value to dynamically adjust the membership functions of the linguistic values of a specific QoS factor. Typically, each set of application specifications requires a custom-made set of fuzzy membership functions and a fuzzy rule base to support the specific application requirements, which are too low-level and fuzzy domain specific for typical application developers to handle. The membership function adaptation mechanism is much simpler for application developers and even users to use to adjust the fuzzy membership functions of various concerned QoS factors to the desired forms.

- **Importance Weight Adaptation** supports ad-hoc modification of the inter-QoS factor prioritization. Based on a similar requirement as in *membership function adaptation*, this mechanism tries to avoid a redesigning

of the fuzzy rule base and fuzzy membership functions when a set of new user requirements has emerged. This is done by dynamically changing the importance weight values of QoS factors of different abstraction levels to reflect the priorities of different QoS factors as specified in the user and application requirements. This mechanism avoids the typical yet significant effort that application developers need to make to implicitly embed the priority preferences into the fuzzy rules. By coupling the *membership function adaptation* that supports the intra-QoS factor adjustment with the *importance weight adaptation* that supports inter-QoS factor prioritization, MobiPADS provides a comprehensive yet simple set of mechanisms for users and applications to dynamically specify their interpretation of QoS factors of different abstraction levels.

- **Computational Reflection** provides Reflective API for applications to inspect (through reflection) and to adapt (through reification) the behavior of the underlying system components, adaptation rules, and actions of MobiPADS. While MobiPADS offers automated and robust mechanisms for context adaptation and service reconfiguration, at times mobile applications are still in the best position to make critical decisions on the operating context and hence the adaptation strategy. Through the meta-level object representation of the hierarchical fuzzy QoS model and service reconfiguration mechanism, a mobile application can access the contextual information, service configuration, and adaptation strategy of the QoS middleware, and modify these entities to obtain optimal service provision during runtime.

7.1 Results

The complete framework has been successfully implemented using Java as the execution platform, which supports a highly portable system to operate across heterogeneous environments. Extensive experiments have been conducted on a PC platform and a PDA platform with the MobiPADS system. These experiments have revealed promising results that demonstrate the effectiveness, robustness, and the ease of leveraging the QoS management framework for new mobile services and applications in a mobile environment.

The results of the experiments in Chapter 5 have clearly demonstrated the effectiveness of *fuzzy controlled adaptation* and *service chaining* in mapping and adapting to variations in QoS parameters under varying contextual environments. Importantly, the adaptation mechanism is able to effectively reconfigure the service chain to maximize the QoS under a dynamic and constrained resources situation. As the resource level improves, the adaptation mechanism is also able to react to this by reconfiguring the service chain to fully utilize the extra resources.

The results of another set of experiments conducted on a PDA device have demonstrated the highly scalable characteristic of MobiPADS with respect to a number of *QoS parameters*, *fuzzy rule*s, *mobilet services*, and *mobilet profiles*. These aspects have demonstrated linear computational complexities and sub-linear memory requirements. Moreover, a typical MobiPADS setup with 32 QoS inputs and 775 fuzzy rules decreased the battery life of the PDA by only 8.4%, which is promising given the benefits of the adaptation capacity of MobiPADS.

The experimental results in Chapter 6 have shown that *membership function adaptation* can easily adjust the forms of the membership functions of a QoS factor to match the interpretation of the user and application on that QoS factor. Subsequently, the adjustment is effectively reflected on the reconfiguration behavior of MobiPADS, while the resultant QoS performance can match the requirements of the user and application. The results also showed that *importance weight adaptation* can effectively adjust the priorities among concerned QoS factors, such that MobiPADS is able to make proper trade-offs among the QoS factors to match the preferences of different users and applications.

7.2 Future work

The nature of progress frequently dictates that a solution to one problem uncovers several new and interesting directions for exploration. In this section, we present several directions for future work that are motivated by our existing work. Many of the directions discussed in this section directly extend our work by addressing the remaining problems. Other directions focus on interesting components within the architecture, and these motivate research in directions that are independent of the dissertation topic.

7.2.1 Contextual Coverage

One of the future directions is to expand the contextual coverage. In this thesis, we have experimented with local QoS parameters only, and it is desirable to extend the system to support external contexts. This involves the management of external contexts, which include discovery, binding, suspension, rebinding, and detachment. On discovering a new external context, the middleware should be able to

autonomously map the contextual changes to the affected QoS factors, new rules, and new mobilets to be migrated to the mobile device. The system will also have to support various handling schemes for orphan fuzzy rules and mobilets when their associated external contexts are unreachable or expired.

7.2.2 Mobilet Service Profile Probing

A mobilet can intentionally or unintentionally affect a QoS factor so that the extent of the effect is uncertain during the design phase, in which case probing during the testing, deployment, and execution phases is required to determine the actual effects of each of its profiles. The probing of service performance is a research topic on its own, yet for simplicity of illustration we have assumed that the probed adjustments for every mobilet profile discussed in this thesis are static, accurate, and up-to-date. It is desirable for the MobiPADS system to provide a unified probing framework, such that a new mobilet service can be easily and accurately profiled. A developer of a mobilet with probed adjustments should implement an *active probing function* that can be invoked when the mobilet is first installed, to determine the initial values of the probed adjustments. A *passive probing function* should also be implemented, which can be invoked when the mobilet is activated for adaptation. This calibrates and updates the values of the probed adjustments adaptively, so that the values can accurately reflect the adjustments to the current operating platform and environment.

7.2.3 Inter-Application Adaptation

This thesis has generally focused on supporting only one mobile application at a time. It is a natural extension to study the impact of providing concurrent service sessions for multiple applications. This will require the introduction of several

new mechanisms. First, another layer of prioritization among all active mobile applications will have to be added. Second, individual fuzzy QoS factor hierarchies are needed for each application. This will give rise to the problem of redundancy, and new techniques will be needed tackle this redundancy. Third, during the reconfiguration of the service chain, MobiPADS will have to consider not only the impact of new mobilet profiles on the application that it serves, but also the impact of the other applications that rely on the shared resources.

7.2.4 Security

Security has not been our focus in the current framework. There are several issues that should be addressed in this area. First, authentication of the mobilets must be guaranteed. One approach is to digitally sign the mobilets to assure their authenticity. Second, the MobiPADS system should exploit the Java security features [Garms01], so that the MobiPADS platform presents a protected environment for a mobilet. Third, a common objection to real-time transcoders is that they are incompatible with encrypted sessions. A service chain is unable to understand an encrypted session because it would have to allow for the decrypting of data in the middle of the connection, which is potentially hazardous to overall security. An interesting research problem is to consider whether an encryption scheme could be developed to permit the services provided by a mobilet without compromising the security of the session.

7.3 Publications

Segments of this research have been presented in the following publications:

- Siu-Nam Chuang and Alvin T.S. Chan, "Dynamic QoS Adaptation for Mobile Middleware", *IEEE Transactions on Software Engineering*, Vol. 34, No. 6, November 2008, pp. 738-752.

- Siu-Nam Chuang and Alvin T.S. Chan, "MobiPADS: A Mobile QoS Middleware based on Hierarchical Fuzzy Control", *Proceedings of the 2006 IEEE International Conference on Fuzzy Systems* (FUZZ-IEEE 2006), 16-21 July 2006, Vancouver, BC, Canada, pp. 2223-2230.

- Siu-Nam Chuang and Alvin T.S. Chan, "Fuzzy Based Mobile QoS management", *Proceedings of the 7th ACM Postgraduate Research Day*, 25 March, 2006, Hong Kong, China.

- Siu-Nam Chuang and Alvin T.S. Chan, "Active Service for Mobile Middleware", *WWW: Internet and Web Information Systems Journal*, Kluwer Academic Publishers, Vol. 8, No. 2, 2005, pp. 127-157.

- Alvin T.S. Chan, Peter Y.H. Wong, Siu-Nam Chuang, "A Context-aware Request Language for Mobile Computing", *Proceedings of the Second International Symposium on Parallel and Distributed Processing and Applications* (ISPA 2004), Lecture Notes on Computer Science, Springer-Verlag, 13-15 December 2004, Hong Kong, China, pp. 529-533.

- Siu-Nam Chuang, Alvin T.S. Chan, Jiannong Cao, Ronnie Cheung, "Actively Deployable Mobile Services for Adaptive Web Access", *IEEE Internet Computing*, Vol. 8, No. 2, 2004, pp. 26-33.

- Alvin T.S. Chan, Siu-Nam Chuang, Jiannong Cao, Hong-Va Leong, "An Event-driven Middleware for Mobile Context Awareness", *The Computer Journal*, Oxford University Press, U.K., Vol. 47, No. 3, 2004, pp. 278-288.

- Alvin T.S. Chan and Siu-Nam Chuang, "MobiPADS: A Reflective Middleware for Context-Aware Computing", *IEEE Transactions on Software Engineering*, Vol. 29, No. 12, December 2003, pp. 1072-1085.

References

[Al-bar99] A. Al-bar and I. Wakeman. "A Survey of Adaptive Applications in Mobile Computing", *ICDCS* 2001, pp. 246-251.

[Angin98] O. Angin, A. T. Campbell, M. E. Kounavis, and R. R.-F. Liao,"The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking", *IEEE Personal Communications Magazine*, Special Issue on Adaptive Mobile Systems, August 1998.

[Bellavista03] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, "Context-Aware Middleware for Resource Management in the Wireless Internet", *IEEE Trans. Software Eng*., 29(12): 1086-1099 (2003)

[Blair01] G.S. Blair, et al. "The Design and Implementation of Open ORB version 2", *IEEE Distributed Systems Online Journal*, 2(6), 2001.

[Bluetooth08] "The Official Bluetooth Wireless Info Site", Web link, http://www.bluetooth.com, 2008.

[Braden97] R. Braden et al, "Resource Reservation Protocol: Version 1 Functional Specification", IETF RFC2205, Sept. 1997.

[Brumitt00] B. Brumitt, B. Meyers, et al. "EasyLiving: Technologies for Intelligent environment", *Handheld and Ubiquitous Computing*, September, 2000.

[Campbell02] A.T. Campbell , J. Gomez , S. Kim , Z. R. Turányi , A. G. Valkó , C. Wan, “Internet Micromobility", *Journal of High Speed Networks*, v.11(3-4), p.177-198, 2002.

[Campbell96] A.T. Campbell, and G. Coulson, "Implementation and Evaluation of the QoS-A Transport System", *Proc. 5th IFIP International Workshop on Protocols for High Speed Networks (PfHSN'96)*, Sophia Antipolis, France, October 1996.

[Capra03] L. Capra, W. Emmerich, and C. Mascolo. "CARISMA: Context-aware reflective middleware system for mobile applications", *IEEE Transactions on Software Engineering*, 29(10): 929–945, 2003.

[Chalmers99] D. Chalmers and M. Sloman, "A survey of Quality of Service in mobile

computing environments", *IEEE Communications Surveys and Tutorials* 2(2), 1999.

[Chan03] Alvin T.S. Chan and Siu-Nam Chuang, "MobiPADS: A Reflective Middleware for Context-Aware Computing", *IEEE Transactions on Software Engineering*, Vol. 29, No. 12, December 2003, pp. 1072-1085.

[Chan04] Alvin T.S. Chan, Peter Y.H. Wong, S.N. Chuang, "A Context-aware Request Language for Mobile Computing", *Proceedings of the Second International Symposium on Parallel and Distributed Processing and Applications* (ISPA 2004), Lecture Notes on Computer Science, Springer-Verlag, 13-15 December 2004, Hong Kong, China, pp. 529-533.

[Chan04-2] Alvin T.S. Chan, S. N. Chuang, J. Cao, H.V. Leong, "An Event-driven Middleware for Mobile Context Awareness", *The Computer Journal*, Oxford University Press, U.K., Vol. 47, No. 3, 2004, pp. 278-288.

[Chemouil95] P. Chemouil, J. Khalfet, M. Lebourges, "A fuzzy control approach for adaptive traffic routing", *IEEE Communications Magazine* 33 (7), 1995, pp. 70-76.

[Cherkasova02] Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat, "EtE: Passive End-to-End Internet Service Performance Monitoring", In *USENIX Conference Proceedings*, 2002.

[Chuang04] S.N. Chuang, Alvin T.S. Chan, J. Cao, R. Cheung, "Actively Deployable Mobile Services for Adaptive Web Access", *IEEE Internet Computing*, Vol. 8, No. 2, 2004. pp. 26-33.

[Chuang05] S.N. Chuang and Alvin T.S. Chan, "Active Service for Mobile Middleware", *WWW: Internet and Web Information Systems Journal*, Kluwer Academic Publishers, Vol. 8, No. 2, 2005, pp. 127-157.

[Chuang06] S.N. Chuang and Alvin T.S. Chan, "MobiPADS: A Mobile QoS Middleware based on Hierarchical Fuzzy Control", *Proceedings of the 2006 IEEE International Conference on Fuzzy Systems* (FUZZ-IEEE 2006), 16-21 July 2006, Vancouver, BC, Canada, pp. 2223-2230.

[Chuang06-2] S.N. Chuang and Alvin T.S. Chan, "Fuzzy Based Mobile QoS management", *Proceedings of the 7th ACM Postgraduate Research Day*, 25 March, 2006, Hong Kong, China.

[Chuang08] S.N. Chuang and Alvin T.S. Chan, "Dynamic QoS Adaptation for Mobile Middleware", *IEEE Transactions on Software Engineering*, Vol. 34, No. 6, November 2008, pp. 738-752.

[Clayton98] P. Clayton, A. Poulton, "Internet quality of service", In *Proceedings of the 1st South African Telecommunications, Networks and Applications Conference* (SATNAC 98), University of Cape Town, South Africa, 1998.

[Cordy92] J. R. Cordy , M. Shukla, "Practical metaprogramming", *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada, November 09-12, 1992.

[Cukier98] M. Cukier, J. Ren, C. Sabnis, W.H. Sanders, D.E. Bakken, M.E. Berman, D.A. Karr, and R. Schantz. "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects", In *Proceedings of the IEEE 17th Symposium on Reliable Distributed Systems*, pp. 245-253, October 1998.

[DAML08] "DAML Services", Web link, http://www.daml.org/services/owl-s/, 2008.

[Davies96] N. Davies, "The impact of mobility on distributed systems platforms", *Proceedings of the IFIP/IEEE Int'l Conf. on Distributed Platforms*, Dresden, Chapman & Hall, 1996, pp. 1825. 1996.

[DiffServ08] "Differentiated Services (diffserv)", Web link, http://www.ietf.org/html.charters/OLD/diffserv-charter.html, 2008.

[Dijkstra76] E. Dijkstra. *A Discipline of Programming*, Prentice Hall, 1976.

[Evans07] J. Evans, C. Filsfils, *Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice*, Morgan Kaufmann, 2007

[Exposito02] E. Exposito, M. Gineste, R. Peyrichou, P. Sénac, M. Diaz , S. Fdida, "XML QoS specification language for enhancing communication services", *Proceedings of the 15th International Conference on Computer Communication*, Mumbai, Maharashtra, India, pp. 76 - 90, 2002.

[Florissi96] P. G. S. Florissi, "QoSME: QoS Management Environment", Ph.D. thesis, Columbia Univ., 1996.

[Frølund98] S. Frølund, J. Koistinen, "QML: A Language for Quality of Service Specification", Software Technology Laboratory, Hewlett-Packard Company,

Report: HPL-98-10, pp. 63, 1998.

[Fukuda97] K. Fukuda, N. Wakamiya, M. Murata, and H. Miyahar, "QoS Mapping between User's Preference and Bandwidth Control for Video Transport", In *Proceeding of the Fifth International Workshop on Quality of Service* (IWQoS'97), New York, NY, May. 1997.

[Fuller96] R. Fuller and C. Carlsson, Fuzzy multiple criteria decision making: Recent developments". *Fuzzy Sets and Systems*, 78(1996) 139-153.

[Garms01] J. Garms. *Professional Java security*. Birmingham, Wrox Press, 2001

[Goyal01] V. K Goyal, "Multiple description coding: Compression meets the network", *IEEE Signal Processing Magazine*, Vol. 8, No. 5, September 2001, pp. 74-93.

[Grace03] P. Grace, and G. Blair. "Interoperating with Services in a Mobile Environment." *Proceedings of ACM/IFIP International Middleware Conference*, Rio de Janeiro, Brazil, June 2003.

[Gu01] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. *An XML-based Quality of Service Enabling Language for the Web*. Technical report, University of Illinois, April 2001.

[Huard97] J.F. Huard and A.A. Lazar. "On QoS Mapping in Multimedia Networks", In 21th IEEE Annual International Computer Software and Application Conference (COMPSAC'97), Washington DC, Aug. 1997.

[Imielinski94] T. Imielinski and B. R. Badrinath, "Mobile wireless computing: challenges in data management", Commun. of the ACM, vol. 37(10), pp. 1828, 1994.

[Jin04] J. Jin, K. Nahrstedt, QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy, *IEEE Multimedia Magazine*, Vol. 11, No. 3, pp. 74-87, July, 2004.

[Julien06] C. Julien and G.-C. Roman. "EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications," *IEEE Transactions on Software Engineering*. 2006.

[Kaashoek97] F. Kaashoek. "Mobile computing with the Rover toolkit." *IEEE Transactions on Computers*, pp. 337-352, March 1997.

[Katz94] R. H. Katz, "Adaptation and Mobility in Wireless Information Systems", *IEEE Personal Communications*, vol. 1(1), pp. 617, 1994.

[Kiczales97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, et al. "Aspect-Oriented Programming." In *Proceedings of European Conference on Object-Oriented Programming*, Springer, LNCS, vol.1241, pp. 220-242, June 1997.

[Koliver02] C. Koliver, K. Nahrstedt, J.M. Farines, J.S. Fraga, A.S. Sandri, "Specification, Mapping and Control for QoS Adaptation", *Real-Time Systems Journal* 23 (2002), pp. 143-174.

[Kuhns99] F. Kuhns, C. O'Ryan, D. C. Schmidt, O. Othman, and J. Parsons."The design and performance of a Pluggable Protocols Framework for Object Request Broker middleware". In *Proceedings of the IFIP Sixth International Workshop on Protocols For High-Speed Networks*, pp. 81-98, Salem, MA, August 1999.

[Lamanna03] D.D. Lamanna, J. Skene, W. Emmerich, "SLAng: a language for defining service level agreements", *Proceedings of the Ninth IEEE Workshop on Future Trends of istributed Computing Systems*, London, UK, 28-30 May 2003, pp. 100-106, 2003.

[Li97] X. Li, S. Paul, P. Pancha and M. Ammar, "Layered Video Multicast with Retransmission ({LVMR}): Evaluation of Error Recovery Schemes", in *Proceedings of the Sixth International Workshop on NOSSDAV*, 1997.

[Li99] B. Li and K. Nahrstedt. "A control-based middleware framework for quality of service adaptations", *IEEE Journal of Selected Areas in Communications*, 17(9), September 1999.

[Liljeberg95] M. Liljeberg, T. Alanko, M. Kojo, H. Laamanen and K. Raatikainen. "Optimizing Word-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach". *Proceeding of the Second International Workshop on Services in Distributed and Networked Environments*, 1995, pp. 132 -139, 1995.

[MBWA08] "IEEE 802.20 Mobile Broadband Wireless Access (MBWA)", Web link, http://grouper.ieee.org/groups/802/20/, 2008.

[Mysaifu08] "Mysaifu JVM", Web link, http://www2s.biglobe.ne.jp/~dat/java/project/jvm/, 2008.

[Nahrstedt95] K. Nahrstedt, J. M. Smith,"The QOS Broker", *IEEE Multimedia*, Vol. 2(1), pp.

53-67, 1995.

[Nahrstedt96] K. Nahrstedt and J. Smith, "Design, Implementation and Experiences of the OMEGA End-Point Architecture", *IEEE Journal on Selected Areas in Communications*, Vol. 2, No. 7, pp. 1263-1279, September 1996.

[Obraczka98] K. Obraczka and G. Gheorghiu, "The performance of a service for network-aware applications", in *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, Welches, Oregon, United States, 1998, pp. 81-91.

[Pahlavan00] K. Pahlavan, P. Krishnamurthy, A. Hatami, M. Ylianttila, J.P. Makela, R.J. Vallstron, "Handoff in hybrid mobile data networks", *IEEE Personal Communications*, Vol 7(2).pp 34-47, 2000.

[Park92] D. Park, Z. Cao, and A. Kandel, "Investigations on the applicability of fuzzy inference," *Fuzzy Sets System*, vol. 49, pp. 151–169, 1992.

[Parlavantzas00] N. Parlavantzas, G. Coulson, M. Clarke, and G. Blair, "Towards a Reflective Component Based Middleware Architecture," *Workshop on Reflection and Metalevel Architectures*, Sophia Antipolis and Cannes, France. June 13, 2000.

[Passino98] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Menlo Park, CA: Addison Wesley Longman, 1998.

[Pitsillides97] A. Pitsillides, Y. A. Sekercioglu, and G. Ramamurthy,"Effective Control of Traffic Flow in ATM Networks using Fuzzy Explicit Rate Marking," *IEEE Journal of Selected Areas in Communications*, Vol. 15, No. 2, 1997, pp. 209-225.

[Qbone08] "QBone Bandwidth Broker Architecture", Web link, http://qbone.internet2.edu/bb/bboutline2.html, 2008.

[Quo08] "Quality Object Project", Web link, http://quo.bbn.com, 2008.

[Raju91] G.V.S. Raju, J. Zhou, R.A. Kisner, "Hierarchical fuzzy control", *Int. J. Control* 54 (5), 1991, pp. 1201-1216.

[Roman01] M. Roman, F. Kon, and R. H. Campbell. "Reflective Middleware: From Your Desk to Your Hand." *IEEE DS Online* (Special Issue on Reflective Middleware), 2001.

[Roman02] M. Roman, C. K. H., R. Cerqueira, A. Ranganathan, Roy H. Campbell, and K.

Nahrstedt. "Gaia: A Middleware Infrastructure to Enable Active Spaces." *IEEE Pervasive Computing*, Oct-Dec, pp.74-83, 2002.

[Ronald98] R. Ronald, "On the construction of hierarchical fuzzy systems models", *IEEE Trans. Systems Man Cybernet*, 28 (1), 1998, pp. 55-66.

[Ross04] T. J. Ross, *Fuzzy Logic with Engineering Applications*. Hoboken, NJ, Wiley, 2004.

[Singh00] R. Singh, A. Ortega, L. Perret, and W. Jiang, "Comparison of multiple description coding and layered coding based on network simulations," *VCIP*, January 2000.

[Smith84] B.C. Smith. "Reflection and semantics in Lisp", In Conference record of POPL, pp. 23-35, 1984.

[TEQUILA08] "IST TEQUILA", Web link, http://www.ist-tequila.org/, 2008.

[Tsang98] D.H.K. Tsang, B. Bensaou, S.T.C. Lam, "Fuzzy-based rate control for real-time MPEG video", *IEEE Transactions on Fuzzy Systems*, Vol. 6, No. 4, November 1998, pp. 504-516.

[UML08] "UML resource page of the Object Management Group", Web link, http://www.uml.org, 2008.

[WiMax08] "The WiMax Forum", Web link, http://www.wimaxforum.org, 2008.

[Yager81] R.R. Yager. "A new methodology for ordinal multiobjective decisions based on fuzzy sets". *Decision Sciences*, Vol. 12, 1981, pp. 589-600.

[Zhou04] C. Zhou, L. Chia, B. Lee, "DAML-QoS Ontology for Web Services", *Proceedings of the IEEE International Conference on Web Services* (ICWS'04), June 6-9, 2004, San Diego, California, USA, pp. 472-479, 2004.