Dynamic Reconfiguration for Complex Multimedia Applications

Baochun Li, Klara Nahrstedt *
Department of Computer Science
University of Illinois at Urbana-Champaign
b-li@cs.uiuc.edu, klara@cs.uiuc.edu

Abstract

Current state-of-the-art distributed multimedia applications require an environment that provides stable Quality of Service (QoS). However, these applications typically run on best-effort heterogeneous platforms, and compete with other applications or connections for end system and network resources, hence suffer from instability and dynamic variations with regards to QoS. In this paper, we present a dynamically reconfigurable middleware control architecture to enhance adaptation awareness of these applications. Our middleware architecture allows for detecting QoS fluctuations in the surrounding environment and signaling optimal control actions to the application. Furthermore, we discuss the design of a core component, the *Configurator*, that adopts a fuzzy control approach to compute optimal control actions. Preliminary experimental results with a distributed visual tracking application show that our approach is viable and effective in controlling adaptive applications.

1 Introduction

Complex distributed multimedia applications typically require the underlying environment to provide a desired level of Quality of Service (QoS). However, in heterogeneous end-to-end environments, QoS-aware system components may coexist with QoS-unaware components along the end-to-end path. If service with statistical guarantees or best-effort service exists in the underlying environment of the applications, the QoS level that the application demands may not be satisfied continuously. The violation of QoS requirements may be caused by physical resource limitations such as inherent bandwidth variations in wireless links, or by statistical multiplexing of a dynamic number of application tasks sharing the same resource pool in end systems and networks. For example, in an end system without real-time prioritized scheduling and reservation mechanisms for CPU resources, CPU-intensive applications may not be able to receive constant QoS with respect to their timeliness requirements in application execution.

The applications, residing on top of the above described environment, must be adaptive to the variations in their end-to-end execution. This means that applications which have strict mission-critical real-time requirements do not fit in this environment, because we cannot provide them with deterministic QoS guarantees that they need. Furthermore, this implies that we need to consider applications which have demands and are flexible within a specific QoS range, such as $[Q_{min}, Q_{max}]$, to allow room for adaptations to occur. In these *flexible* applications, adaptations play an important role because constant guarantees are either not possible, when physical limitations are present, or not cost-effective, when it is impossible to predict the maximum QoS requirements to be reserved for interactive applications.

In general, flexible applications demonstrate several *scalability* properties. First, they can accept and tolerate resource scarcity to a certain minimum bound Q_{min}, and can improve its performance if given a larger share of resources. Second, they are willing to sacrifice the performance of some quality parameters in order to preserve the quality of critical parameters. When QoS variations occur and QoS cannot be maintained for all application quality parameters, it is possible and desirable to trade off less critical parameters for preserving quality assurance of critical parameters.

*This research was supported by the Air Force Grant under contract number F30602-97-2-0121, NASA Grant under contract number NASA NAG 2-1250, and National Science Foundation Career Grant under contract number NSF CCR 96-23867.

The objective of our work is to support QoS adaptations in flexible applications with several middleware components. The services provided by these components have three main goals. First, they serve as a centralized global coordinator to control the adaptation behavior of all concurrent application tasks in the end system, so that if viewed globally, these applications do not adapt in a conflicting or unfair way. Second, they enhance the adaptation awareness of flexible applications, by making decisions to control their adaptive behavior. The adaptation awareness includes when, how and to what extent adaptation is carried out in the applications. Third, they serve as an observer to monitor the dynamics in the environment, so that informed decisions can be made to control the applications.

In order to balance between the centralized approach of making optimized and fair control decisions and the diversely different QoS requirements from different applications, we consider two separate middleware components: Adaptor and Configurator. The *Adaptor* makes control decisions with global awareness of application requirements and resource availability of the entire system, while the *Configurator* is in charge of providing translation and configuration services. The Configurator translates the normalized control decisions, generated by the Adaptor, into the actual parameter-tuning actions or reconfiguration choices used during the execution of the application. Note that this translation mechanism differs and supersedes the QoS translation between different categories of QoS parameters in the way that it translates the control actions, rather than parameter values.

In our previous work [12], we exploited the analogy between a control system and the adaptation behavior, and developed a Task Control Model that gives theoretical results to reason about and prove stability, fairness, and adaptation agility properties of the adaptation behavior. The *Adaptor* uses the Task Control Model and the results developed in [12] to make control decisions for individual target applications with a global awareness of resource availability. The complementary work, presented in this paper, focuses on the internal algorithms used in the *Configurator*, which allow for a single control policy to control applications via different application-specific mechanisms.

The major contributions of this paper are the following. (1) We adopt a fuzzy control approach in the design of the Configurator, so that the adaptation choices and preferences for different applications can be expressed explicitly in a *rule base* and *member functions* for each linguistic value. The *rule base* provides linguistic rules that the *inference engine* is based on, and the *inference engine* generates manipulating signals that control the actual application. (2) We show that our approach is feasible when dealing with nonlinearities of different control choices, such as a hybrid combination of parameter-tuning actions and reconfiguration choices. These choices are naturally nonlinear and mostly discrete, while the rules that guide the decision-making process are mostly intuitive and heuristic. These rules are application-specific and determined by human experts who use trial-and-error approaches to best fit the interests of the particular application. (3) We validate our approach with a complex client-server based visual tracking application. This application is flexible and includes a rich set of adaptation possibilities when running in QoS-unaware environment.

The rest of this paper is organized as follows. In Section 2, we discuss existing work related to our approach. In Section 3, we give an overview of the middleware control architecture. In Section 4, we review our Task Control Model proposed in our previous work [12], designed for the Adaptor, in the context of a distributed visual tracking application. In Section 5, we focus on our fuzzy control approach to design the Configurator, including the specification of application-specific preferences, the benefits of adopting the approach, and a detailed analysis in the context of the distributed visual tracking application with respect to its parameter-tuning and reconfiguration options. In Section 6 we present preliminary results with the visual tracking application. Section 7 concludes the paper and discusses future work.

2 Related Work

The field of QoS Adaptation in distributed multimedia applications has been studied by various previous work. In [13], a graceful adaptation service dynamically manages the QoS of real-time communications by changing the parameter configurations in the network with no or limited disruption. It is implemented using mechanisms such as dynamic re-routing and load balancing. In contrast to our work, it focuses primarily on communication subsystems, while our approach focus on adaptation within user-level applications.

Other work [1] [2] [20] tries to adapt frame, layering or coding parameters in multimedia flows so that output rate can be varied according to feedback from the network. These schemes propose mechanisms for graceful degradation with multimedia data flows, and they do not address the problem of deciding the timing, scale and choices of adaptation actions. In [19], the authors proposed a framework for the communication subsystem to provide flexible best-suited services to the applications with different functional features and numerical QoS attributes. Our framework controls

the applications to adapt, rather than the communication subsystems. In [7], the authors proposed adaptation at the configuration level, which carries out transparent transition from primary components to alternative components, as well as at the component level, which redistributes resources in different components so that a QoS tradeoff can be made. Similarly, our approach also models applications as different tasks. However, we differ from the previous work in the sense that we propose algorithms to make choices on adaptation timing, scale and methods used, which balances between the frequency and responsiveness of adaptation actions within the application.

The application of control theories has been explored in recent work in the area of QoS adaptation. In one paper [18], the application of control theory is suggested as a future research direction to analyze adaptation behavior in wireless environments. In another [3], a control model is proposed for adaptive QoS specification in an end-to-end scenario. In the third example [6], the time variations along the transmission path of a telerobotics system are modeled as disturbances in the proposed perturbed plant model, in which the mobile robot is the target to be controlled. In our previous work [12], theoretical proofs are given for various properties applying control theory to model QoS adaptation.

Our work is also closely related to and utilizes the knowledge of dynamic resource allocations. In [9], the global resource management system that relies on middleware services as agents to assist resource management and negotiations. In [15], the work focuses on maximizing the utility functions, while keeping QoS received by each application within a feasible range. In [17], the authors focus on a multi-machine environment running a single complex application, and the objective is to dynamically change the configuration of the application to adapt to the environment. In comparison, our work focuses on the analysis of the actual adaptation choices, rather than individual or overall utility factors. We also focus on an environment with multiple applications competing for a pool of shared resources, which we believe is a common scenario easily found in many actual systems.

Rich features in the field of fuzzy control systems are also utilized in previous research related to adaptive systems and flow control. In the AutoPilot [16] project, a fuzzy logic approach is adopted to design actuators that process sensory data observed from high-performance parallel programs, so that optimal performance can be achieved by adjusting system parameters, such as those in a parallel I/O file system. The actuators and sensors are functionally similar to the Adaptation Tasks and Observation Tasks in our Adaptors. However, the objectives and domain of operations are notably different. In [14], a fuzzy control approach is used for the purpose of flow control in ATM networks, with linguistic variables being the queue length and the change rate of queue length in each ATM switch. In contrast, our approach focuses on generating control actions to control distributed multimedia applications themselves, with efforts to best adapt to the environment.

3 The Middleware Control Architecture for Flexible Applications

A major objective of the middleware control architecture is to observe the current conditions in the distributed environment and signal control actions to the complex distributed applications, so that reconfigurations or parameter-tuning actions are carried out within the application. The architecture consists of two parts, the *Adaptors* and *Configurators*. In an end system, each Adaptor corresponds to a single type of resource, such as CPU or transmission bandwidth, and consists of an *Adaptation Task* and an *Observation Task*. Each Configurator corresponds to a single Target Task or application, and makes control decisions based on the output of several Adaptors related to several types of resources. The interaction between different middleware components and the application is through a specific service enabling platform, with the current implementation being CORBA. Figure 1 shows an overview of the architecture.

In Section 4, we review our design of middleware Adaptors using the Task Control Model. In Section 5, we develop our design of middleware Configurators using the Fuzzy Control Model. We discuss our approach in the context of the client-server based visual tracking application, which we show preliminary experimental results in Section 6.

4 The Task Control Model for Designing Middleware Adaptors

In order to control the adaptive behavior of distributed applications with a global awareness of resource availability, so that a fairness property can be achieved, we integrated a *Task Control Model* as proposed in our previous work [12] into the design of our middleware *Adaptors*. This is complementary to the design of Configurators, which is the focus of this paper. The Adaptors promote global awareness, while Configurators focus on application-specific nonlinear

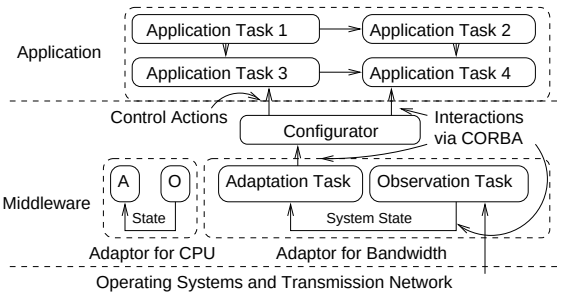


Figure 1: An Overview of the Middleware Control Architecture

adaptation possibilities. We review the Task Control Model briefly in the context of a real world application, the client-server based visual tracking application.

4.1 The Task Control Model

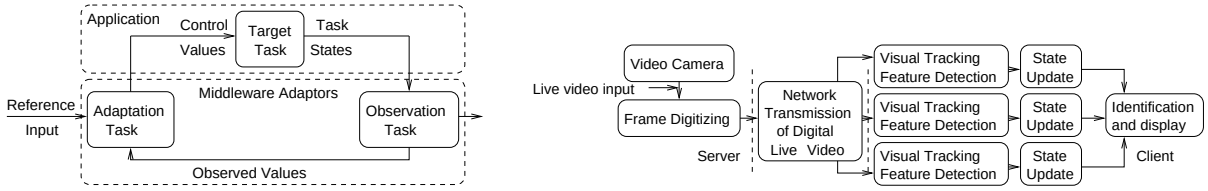
Before proceeding to analyze quantitative properties of the adaptive behavior, we present the Task Control Model to model the execution environment of flexible distributed applications in the paradigm of traditional control systems. For this purpose, we consider each distributed application as an ensemble of functional components, which we refer to as *tasks*. Tasks are execution units that perform certain actions to deliver results. All tasks in an application can be presented as a directed acyclic graph, which illustrates the producer-consumer dependency among tasks. A directed edge from task T_i to T_j indicates that T_j uses the output produced by T_i. Each task can be uniquely characterized by its *input quality*, *output quality* and *utilized resources* [10].

The Task Control Model focuses on one task in the directed acyclic graph, the *Target Task*. In addition, in order to utilize the analogy with control systems, we introduce an *Adaptation Task*, which enforces the control policy, as well as an *Observation Task*, which observes or estimates the states of the Target Task and feeds them back to the Adaptation Task. These tasks are embedded in the functional middleware component of an end system, namely, the Adaptor, as shown in Figure 2(a). By coordinating with a middleware Configurator, the middleware Adaptor effectively controls the Target Task in the application, and assures that the output quality of critical quality parameters is preserved within the desired QoS level, regardless of variations in resource availability.

As a proof-of-concept system, we present a client-server based visual tracking application that validates our approach. The basic operations of the application are described as follows. A tracking server obtains live video feed from an online video camera, and sends the video feed over the transmission network to the tracking client. The client executes one or more CPU-intensive kernel tracking algorithms, which identify and track objects of interest to the user. The result is presented to the user visually showing coordinates of the tracked objects. Naturally, the critical quality parameter in this application is *tracking precision*. An example of the Task Control Model presented in the form of a directed acyclic graph is shown in Figure 2(a), and an example for the distributed visual tracking application is shown in Figure 2(b).

The visual tracking application is flexible in the following manner. First, since network bandwidth between the tracking server and the client may fluctuate, the image quality and delivery timeliness may be affected, thus affecting tracking precision. The application allows for degradation to lower image resolutions or smaller image sizes and still be able to preserve tracking precision. Second, since the tracking algorithms can track multiple objects simultaneously, the available CPU cycles may not be sufficient for all objects. The application allows for degradation by tracking only the most important objects. It may also replace a more CPU-intensive tracking algorithm with a less intensive one, in order to keep the tracking precision. Third, the application may wish to reconfigure itself and add compression or security modules in order to take advantage of excessive CPU cycles and release the burden on transmission bandwidth. The application shows its superior flexibility by the above parameter-tuning and reconfiguration choices.

In the middleware Adaptor, our objective is to adequately model the Target Task, and utilize the Task Control Model to generate appropriate control signals, which are directed to the Configurator and translated into control actions within the application.



(a) Task Control Model Implemented in the Middleware Adaptor

(b) A Client-Server Based Visual Tracking Application

Figure 2: The Middleware Adaptors in the Context of a Distributed Visual Tracking Application

4.2 A Linear Model for the Target Task

In order to develop control algorithms in the Adaptation Task, we need to have a precise analytical model to characterize the internal dynamics in the Target Task. The independent parameters in this model are referred to as *task states*. Let us assume that $\mathbf{x}$ denotes task states, $\mathbf{u}$ denotes controlled input, $\mathbf{y}$ denotes system output, $\mathbf{w}$ denotes system noise, $\mathbf{z}$ denotes observation, and $\mathbf{v}$ denotes observation error. We use linear and discrete-time models to model a Target Task as follows:

$$\mathbf{x}(k) = \mathbf{\Phi}\mathbf{x}(k-1) + \mathbf{\Gamma}\mathbf{u}(k-1) + \mathbf{w}(k-1) \tag{1}$$

$$\mathbf{y}(k) = \mathbf{H}\mathbf{x}(k) \tag{2}$$

$$\mathbf{z}(k) = \mathbf{y}(k) + \mathbf{v}(k) \tag{3}$$

where $k = 1, \ldots, k_{max}$ with k_{max} as the maximum possible time instant, and $\mathbf{\Phi}$, $\mathbf{\Gamma}$, and $\mathbf{H}$ are known matrices without error.

We illustrate the above discussed generic models on a concrete example by considering the following scenario. We assume multiple tasks competing for a shared resource pool with the capacity C_{max}. Each task T_i makes *new requests* r_i for resources in order to perform their actions on inputs and produce outputs. These requests may be *granted* or *outstanding*. If a request is granted, resources are allocated immediately. Otherwise, the request waits with an outstanding status until it is granted. The system grants requests from multiple tasks with a constant *request granting rate* y.

For different types of resources, the notation *resource requests* is interpreted differently. For temporal resources, such as communication bandwidth and CPU, where the resources are shared in a temporal fashion, outstanding resource requests are mapped to data in the waiting queue, and granted requests are mapped to allocated temporal resources, such as bandwidth. For example, for transmission tasks, the request granting rate y denotes the total physical bandwidth of the communication channel, while the granted requests denote data that have completed transmission over the channel, and outstanding requests denote data that are in flight in the channel or in the waiting queue.

Figure 3 illustrates the above scenario for the Target Task, along with an Adaptation Task implemented in the middleware Adaptor. As the figure shows, the Target Task includes functions which request new resources, as well as functions which wait, because their resource requests are outstanding due to unavailability of resources. The Adaptation Task computes a new request rate and issues it to the Target Task as the *throttled request rate* value.

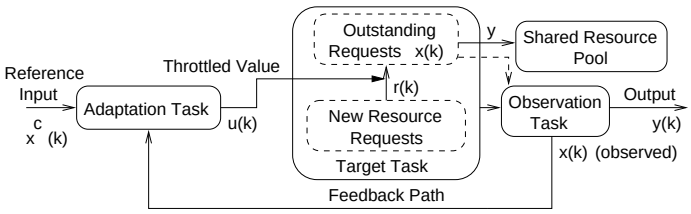


Figure 3: A Linear Model for the Target Tasks

In this particular scenario, we can use a linear control model within the Adaptation Task to adapt the Target Task fairly.

We define the following terms for a *Target Task* T_i:

1. t_c is a constant sampling time interval, which represents the time elapsed in the interval $[k, k+1]$, k being time instants satisfying $k = 1, \ldots, k_{max}$;
2. $y_i(k)$ is the number of granted requests for T_i in the interval [k, k+1], and the observation of which is $z_i(k)$ with an error of $v_i(k)$;
3. y is the total number of granted requests for all tasks, which we assume to be a constant;
4. $u_i(k)$ is the number of throttled resource requests in $[k, k+1]$ controlled by *Adaptation Task* for T_i;
5. $x_i(k)$ is the number of outstanding resource requests made by T_i;
6. $x(k)$ is the *total number* of *outstanding resource requests* made by *all tasks* at time k;

With these notations, we may define the following to model the Target Task:

$$\dot{x} = x(k) - x(k-1) = \sum_{i=0}^{M(k-1)} u_i(k-1) - y \tag{4}$$

where $M(k)$ is the total number of active tasks competing for resources in the system in $[k, k+1]$.

The difference equation (4) depicts the internal dynamics of the Target Task. Intuitively, the difference between outstanding resource requests of two adjacent time instants should be equivalent to the difference between the input resource requests and the granted (output) resource requests. In our scenario, with the presence of the Adaptation Task, the input resource request rate is the same as the number of *throttled* resource requests $u_i(k)$, which is the generated control values by the Adaptation Task.

4.3 Control Algorithms in Adaptation Tasks

In [12] we developed a standard proportional-integral- derivative (PID) control [5][1] as an example of the control algorithms embedded in the Adaptation Task. In this case, $u_i(k)$ obeys the equation

$$u_i(k) = u_i(k-1) + \alpha[x^c(k) - x(k)] + \beta\{[x^c(k) - x(k)] - [x^c(k-1) - x(k-1)]\} \tag{5}$$

where α and β are configurable scaling factors. We were able to prove the following properties given priority weights for each task:

- *Equilibrium:* The number of outstanding resource requests x in the system, established by Equation (4) and (5), will converge to an equilibrium value which equals to the reference value x^c.
- *Fairness:* The system fairly shares resources among competing tasks according to the weighted max-min fairness property, based on their priority weights.
- *Stability:* There exist appropriate values of parameters α and β so that all tasks in the system are asymptotically stable around a local neighborhood, for any pre-determined priority weight for a task T_i.

5 The Fuzzy Control Model for Designing Middleware Configurators

Within the Adaptation Tasks implemented in the middleware Adaptors, global fairness, responsiveness and stability properties are ensured by the well-designed control algorithms, and reflected by the theoretical control values expressed in the form of the number of *throttled resource requests* $u(k)$ in the time interval $[k, k+1]$. This section focuses on the design of middleware *Configurators*, which determine nonlinear and discrete control actions based on application-specific needs, as well as control values from the Adaptors.

[1] PID control is a classic control algorithm where the control signal is a linear combination of the error, the time integral of the error, and the rate of change of the error.

5.1 Motivations behind the Fuzzy Control Model

Similar to the role of the Task Control Model in the design of middleware Adaptors, we utilize the rich semantics and features in existing fuzzy logic and fuzzy control systems theory to design the control algorithms in the middleware Configurators. This design model for the Configurators is referred to as the *Fuzzy Control Model*.

The advantages of adopting the Fuzzy Control Model are the following:

1. Taken the fact that multiple reconfiguration options and parameter-tuning possibilities exist in a typical complex application, the controllable regions and variables within the application are in most cases discrete, non-linear and complex. In these applications, the models for the Target Tasks are nonlinear in nature. On the other hand, a fuzzy logic control system can be conceived as a nonlinear control system, in which the relationships between inputs and control outputs are expressed by using a small number of *linguistic rules* stored in a *rule base*. The nonlinearity of the fuzzy controller matches naturally with the nonlinearity of controllable regions and adaptation possibilities within an application.

2. The Fuzzy Control Model is inherently generic and highly configurable. Both the rule base and the definition of membership functions for linguistic values can be configured to be application-specific. The Fuzzy Control Model offers a common design for Configurators suitable for all applications, without loss of generality and configurability.

3. The Fuzzy Control Model includes the fuzzy inference engine (with its linguistic rule base), which represents the decision-making process and resembles natural human communication and reasoning. For this reason, it is natural and straightforward for the application to specify its own adaptation preferences and decisions in the form of linguistic values and rules. The merits of the simplicity, however, do not affect the flexibility and power of fuzzy control systems to define the most complicated nonlinear multiple-dimensional control surface.

5.2 Configurator Design: The Fuzzy Control Model

As stated in previous sections, the Configurator takes the output of the Adaptor as input, and generates actual manipulating signals as control actions to command reconfiguration or parameter tuning within the application. The Fuzzy Control Model, shown by Figure 4, is used for designing the overall architecture of the Configurator. The model comprises of five components built within the Configurator. The *fuzzy inference engine* implements particular fuzzy control algorithms defined in the application-specific *rule base* and *membership functions* for linguistic values. The *input normalizer*, *fuzzifier* and *defuzzifier* prepare input values for the fuzzy inference engine, and convert fuzzy sets (the decisions made by the inference engine) to the actual real-world control actions for the applications. Note that the Fuzzy Control Model is utilized as a nonlinear transfer element in the overall control loop, which means that the rule based representation of the model does not include any dynamics with respect to time, e.g. derivation or integration.

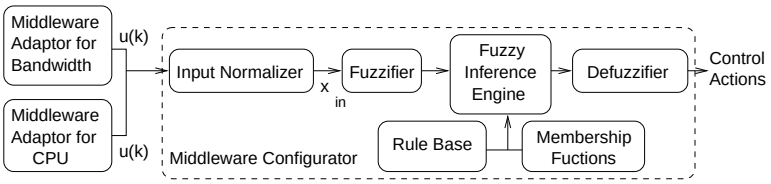


Figure 4: The Overall Architecture of the Fuzzy Control Model

While the architecture of the Fuzzy Control Model is generic and can be applied to any applications by configuring the rule base and membership function definitions, we adopt the client-server based visual tracking application as a concrete example to elaborate our design of the Configurator.

5.2.1 The Design of Rule Base

The decisions of selecting linguistic values and rules in the rule base are based on a combination of human expertise and trial-and-error experiments on the particular application. The tradeoff is to decide on a minimum number of linguistic rules, while still maintaining the desired accuracy to achieve an acceptable adaptation performance. All of

the linguistic values used in the rule base should use words of a natural or synthetic language, such as `moderate` or `below_average` for the linguistic variable `cpu_demand`. These values are modeled by fuzzy sets. In most cases, this form of representation leads to compact description of the adaptation behavior within the application.

The design of the rule base is a two-phase process. First, the linguistic rules are determined. Second, membership functions of the linguistic values are set. In the fuzzy control system, the first phase of design generates a set of conditional statements in the form of if-then rules. The generic form is:

$$\begin{aligned} R^{(1)}: &\quad \text{if } X_1 \text{ is } A_1^{(1)} \text{ and } \ldots \text{ and } X_n \text{ is } A_n^{(1)} \text{ then } Y \text{ is } B^{(1)} \\ \ldots & \qquad\qquad \ldots \\ R^{(m)}: &\quad \text{if } X_1 \text{ is } A_1^{(m)} \text{ and } \ldots \text{ and } X_n \text{ is } A_n^{(m)} \text{ then } Y \text{ is } B^{(m)} \end{aligned} \tag{6}$$

where $X_1 \ldots X_n$ and Y are linguistic variables, $A_1^{(k)} \ldots A_n^{(k)}$ and $B^{(k)}$ $(k = 1, \ldots, m)$ are linguistic values, defined by fuzzy sets $\tilde{A}_1^{(k)} \ldots \tilde{A}_n^{(k)}$ and $\tilde{B}^{(k)}$ $(k = 1, \ldots, m)$, respectively. These linguistic values are also characterized by their membership functions, $\mu_{A_l^{(k)}}(x)$ and $\mu_{B^{(k)}}(y)$ $(l = 1, \ldots, n)$, respectively, with x and y being the elements of universal sets U and V. Each rule defines a fuzzy implication that performs a mapping from fuzzy input state-space to a fuzzy output value. After the defuzzification process, the fuzzy output value directly corresponds to a particular control action within the application.

The fuzzy inference engine operates by using the dual concepts of generalized modus ponens and compositional rule of inference [4]. For mathematical completeness of the paper, the internal mechanisms of the inference engine are covered in Appendix A. In our preliminary implementation of the fuzzy inference engine, we adopt the C-FLIE inference engine implementation [14] as well as its input format for specifying the rule base and membership functions.

We consider the visual tracking application as an example for designing the rule base and membership functions used in the middleware Configurator. As we noted, the ultimate objective and most critical application-specific quality parameter in the application is *tracking precision*. If the precision is compromised, the objects lose track and other parameters are not meaningful.

In this application, the adaptation possibilities can be classified into two categories. First, control actions may occur in order to adapt to transmission bandwidth variations, so that bandwidth requirements within the application are adjusted to maintain tracking precision. Second, adaptations may take place to adapt to varied availability of CPU cycles, so that CPU requirements are adjusted. For other complicated applications, memory requirements or storage I/O requirements are also taken into consideration.

Within the above categories of adaptation possibilities, there are two different kinds of adaptation actions. First, parameter-tuning actions try to tune quantitatively continuous parameters, such as image size, to meet adaptation goals. Second, reconfiguration possibilities within the application enable adaptation by selecting among different configuration options, each having diverse requirements for resources. This process sometimes involves an alteration in the Task Flow Graph of the application.

Divided in two major categories, we have identified the adaptation possibilities in this application as the following.

- **Adaptation of Communication Bandwidth Requirements**. Since the application is client-server based, sufficient bandwidth is required for preserving tracking precision. First, the following options exist for parameter-tuning actions for uncompressed image transfer. (1) The *image size* can be enlarged or reduced to adjust bandwidth requirements, by *chopping* the edges. The tradeoff is that the smaller the image, the higher the probability that the objects move out of range. (2) The *image size* can be enlarged or reduced by *scaling*, the tradeoff being a higher CPU load for real-time per-frame scaling. (3) The *color depth* can be altered. Existing choices for coding one pixel are 24 bits RGB, 16 bits packed RGB, 8 bits grayscale or 1 bit black-and-white. Second, if we consider reconfiguration choices, *compression* and corresponding *decompression* can be activated, using available choices such as Motion-JPEG and streaming MPEG-2 among others. Bandwidth requirements are reduced dramatically at the expense of increased CPU load.

- **Adaptation of CPU Requirements**. The tracking algorithms are inherently computationally intensive. In the current implementation, there are three frequently used tracking algorithms. *Line* tracking and *corner* tracking are edge based algorithms, *SSD* tracking is a region based algorithm. Table 1 shows that these algorithms present diverse computational requirements. In addition, the application can run multiple algorithms tracking multiple objects simultaneously, with each algorithm referred to as a *tracker*, and the tradeoff being increased

computation load. These facts motivate the following reconfiguration choices: (1) Add additional trackers to utilize idle CPU; (2) Drop running trackers to decrease CPU demand; (3) Replace existing trackers by less or more computationally intensive trackers. Finally, parameter-tuning adaptation may also be applied by modifying the size of the *tracked region* of a specific tracker, effectively tuning the computational load of the tracker. The tracked region is defined as the searching range of the tracker in the feature detection stage of computation.

Computation Stage	*Line Tracking*	*Corner Tracking*	*SSD Tracking*
Initialization: Average	0	0	15
Feature Detection: Average	171.37	195.73	135.8
Feature Detection: Variance	539.90	331.37	471.41
Internal State Update: Average	0	0	1.07
Internal State Update: Variance	0	0	16.48
Display: Average	0	1.57	7.83
Display: Variance	0	22.87	63.59

Table 1: Computational Load of Tracking Algorithms: A Comparison (milliseconds)

The adaptation measures described above make it possible to design the rule base for the visual tracking application, following the generic form given in Equation (6). As Figure 4 shows, the fuzzy control model takes the output of multiple Adaptors as input, each of which corresponding to one type of resource. In the particular case of visual tracking, we focus on two types of resources: CPU cycles and transmission bandwidth. In our rule base, the linguistic variable `cpu` corresponds to the values $u(k)$ generated by the Adaptor observing the CPU resource, and the linguistic variable `rate` corresponds to the values $u(k)$ generated by the Adaptor observing transmission bandwidth. The range of measuring linguistic variable `cpu` is $[0, 1000]$ with an unit of milliseconds of CPU required per second, and the range of measuring linguistic variable `rate` is $[0, 2000]$ with an unit of kilobytes transmitted per second. Before processing in the inference engine, the numerical crisp values u_k are first linearly normalized to the above ranges and units, then mapped to a fuzzy set by the fuzzification process, of which the mathematical details are documented in Appendix B.

There are two inference outputs using the rule base, corresponding to the bandwidth adaptation and CPU adaptation measures, respectively. The linguistic variables used are `rate_demand` and `cpu_demand`, respectively. The linguistic values used for both `cpu` and `rate` are `very_low`, `below_average`, `moderate`, `above_average` and `very_high`.

We present one design of the rule base using the input format in the C-FLIE implementation of fuzzy inference engine.

```
/* linguistic rules corresponding to bandwidth adaptation */
if rate is very_high then rate_demand is chopped_image
if cpu is very_high and rate is below_average then rate_demand is compress
if cpu is very_high and rate is very_low then rate_demand is compress
if cpu is above_average and rate is below_average then rate_demand is compress
if cpu is moderate and rate is moderate then rate_demand is scaled_image
if cpu is below_average and rate is above_average then rate_demand is RGB24_color
if cpu is below_average and rate is moderate then rate_demand is RGB16_color
if cpu is below_average and rate is below_average then rate_demand is grayscale
if cpu is very_low and rate is very_low then rate_demand is back_and_white

/* linguistic rules corresponding to cpu adaptation */
if cpu is very_high and rate is above_average then cpu_demand is add_tracker
if cpu is very_high and rate is very_high then cpu_demand is add_tracker
if cpu is below_average and rate is very_low then cpu_demand is drop_tracker
if cpu is very_low and rate is very_low then cpu_demand is drop_tracker
if cpu is moderate and rate is moderate then cpu_demand is replace_tracker
if cpu is above_average and rate is above_average then cpu_demand is adjust_region
```

5.2.2 The Design of Membership Functions

In normal design practices of fuzzy control systems, Gaussian, triangular or trapezoidal shaped membership functions are used to define the linguistic values of a fuzzy variable. Since triangular and trapezoidal shaped functions offer more computational simplicity, we choose them to define all membership functions for linguistic values used in the rule base.

The particular design of these membership functions is largely application-specific. In our visual tracking application, we have defined the membership functions as shown in Figure 5, in four universal sets for variables `cpu`, `rate`, `cpu_demand` and `rate_demand`, respectively.

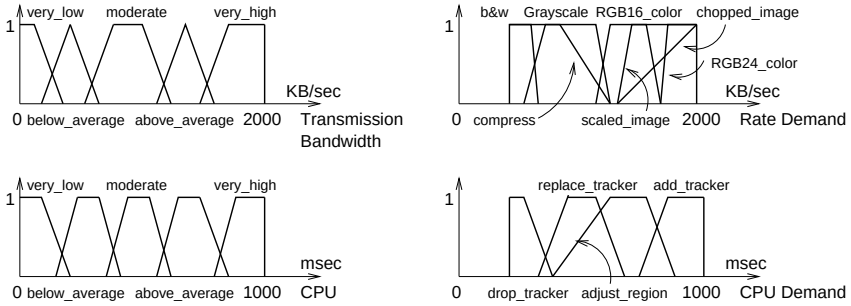


Figure 5: Membership Functions of the Linguistic Values

5.2.3 The Defuzzification Process

Since the decision of the inference engine is expressed in fuzzy sets, in order to be able to use it as a control signal for applications, it has to be mapped to reconfiguration options or crisp numerical values of parameter-tuning actions. The defuzzification process produces a non-fuzzy output, y_{out}, whose objective is to represent the possibility distribution of the inference. There is no single method for performing the defuzzification. In fuzzy control systems, because of the ability of generating smoother control surfaces, the *Center of Gravity* method is frequently used. Detailed mathematical definition of the *Center of Gravity* method is documented in Appendix C.

Once y_{out} is obtained, the mappings to the actual control actions are straightforward. If $\tilde{B}$ is a fuzzy set corresponding to a reconfiguration option (e.g. `drop_tracker`, etc.) and $\mu_B(y_{out}) \neq 0$, the corresponding reconfiguration is activated. Otherwise, if $\tilde{B}$ is a fuzzy set corresponding to a parameter-tuning action associated with the parameter p (e.g. `scaled_image` associated with *image size*) and the tuning range $[p_{min}, p_{max}]$, then the modified value of p is set at

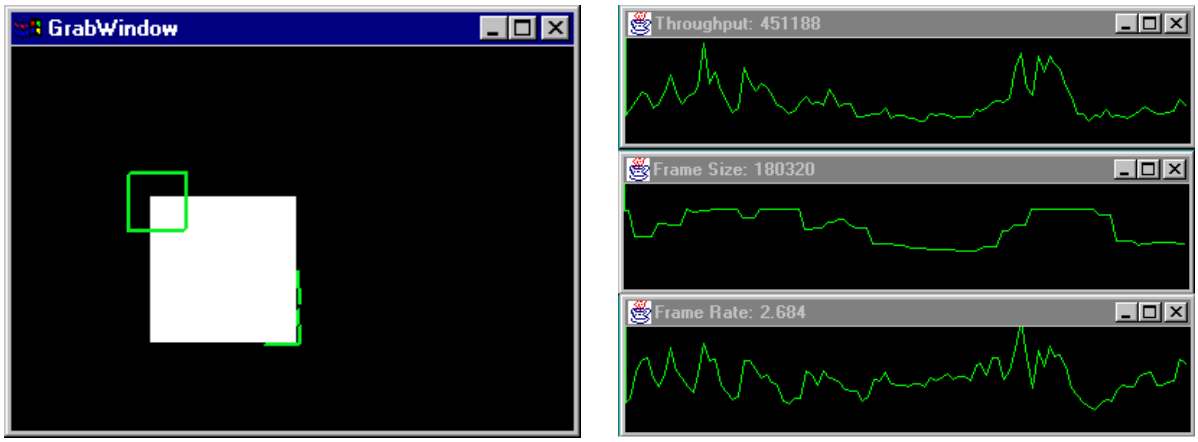
$$p = (p_{max} - p_{min}) * \mu_B(y_{out}) + p_{min} \tag{7}$$

when $\mu_B(y_{out}) \neq 0$.

6 Experiments with the Visual Tracking Application

Based on the design for middleware Adaptors and Configurators presented in previous sections, we have implemented a middleware framework to control the client-server based visual tracking application, which is the example throughout the paper. Based on tracking algorithms implemented in the XVision [8] project, we have successfully implemented this application on the Windows NT 4.0 platform in Visual C++ 5.0, using Windows Sockets 2 API for the network transmission.

Programmed in C++ and Java as middleware components, we implemented the Adaptor, including the Adaptation Task and Observation Task, as well as the internal mechanisms of the Configurator using the Fuzzy Control Model. We adopted the C-FILE implementation [14] as our fuzzy inference engine. All middleware components interact among one another and with the application using service enabling platforms such as CORBA. We use ORBacus 2.0.4 [11] as our CORBA implementation. Figure 6(a) shows the main tracking window of the application, and Figure 6(b) shows the Observation Task running within the middleware Adaptor.



(a) The Main Tracking Window

(b) The Observation Task within the middleware Adaptor

Figure 6: A Running Client-Server Based Visual Tracking Application

As a first series of experiments we tested our system in a varying network environment, in order to explore adaptation possibilities on transmission bandwidth requirements. The network environment is currently a simple network simulator which allows us to simulate bandwidth fluctuations in a typical distributed environment over WAN. The simulator simulates packet delay through a transmission path of multiple network routers, each of them implementing the FIFO scheduling algorithm. Because of the bursty nature of cross traffic, throughput fluctuations may occur at various times over the connection.

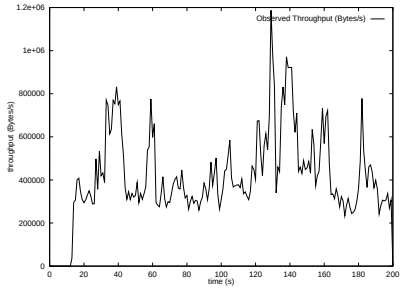
For the purpose of repeating the same set of experiments and for measurements of tracking precision, we use a computer generated image sequence, in which the object moves at fixed speed and path. For the experimental results shown in Figure 7, the moving speed of the rectangle is set at a constant 3 pixels per second continuously. In addition, we assume there are no other CPU intensive process running in the background on the same platform. This is for the purpose of separating the experiments on bandwidth requirements from those on CPU requirements.

In Figure 7, for comparison purpose, the three graphs on the left show the tracking results without any adaptation. Those on the right show the case with adaptation support from the middleware framework, adopting both Task Control and Fuzzy Control Models presented in the previous section. We can observe that by chopping image sizes being transmitted, the bandwidth requirements are effectively reduced, the tracking precision will be preserved without any tracking error at all times during the connection. In contrast, without any adaptation, when the network throughput degrades to a certain degree, the tracking algorithm is not able to keep track of the object, the error accumulates rapidly showing that the tracking algorithm loses the object. This prove-of-concept system validates that the adaptation measures activated by the Configurator are effective in preserving tracking precision in a distributed environment with varying bandwidth. We also observed that the parameter-tuning measures related to image sizes are only effective within a higher range of bandwidth availability. When bandwidth availability becomes even lower, other measures, such as altering color depth and reconfiguring to add compression and decompression modules, are desirable. We are in the progress of adding more adaptation measures within the visual tracking application, so that it may react appropriately to the control signals generated by the middleware Configurator.

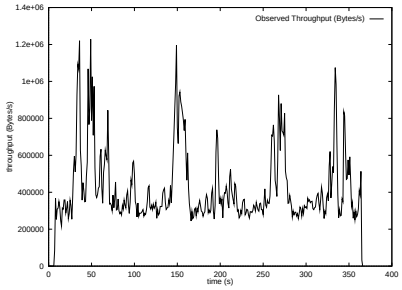
7 Conclusions

In this work, we focused on flexible distributed multimedia applications that need to adapt their behavior to variations of the resource availability and assure quality of critical QoS parameters. In this work we presented the design of the middleware Configurator, which maps numerical values from the middleware Adaptors to actual control actions of parameter tuning or reconfiguration choices. A Fuzzy Control Model was adopted in the design of the Configurator, and the design of the rule base and membership functions was shown in the context of a distributed visual tracking ap-

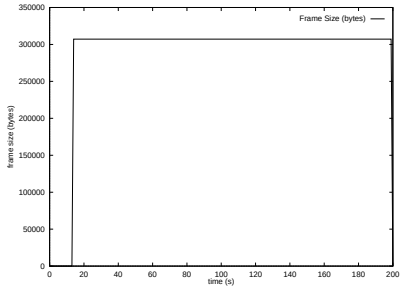
plication. In our preliminary experiments we showed that our model successfully controls the communication aspects of the visual tracking application, and adapts to varying bandwidth in a distributed environment. Ongoing and future work involves extensions of our preliminary experiments, as well as collaboration issues involving multiple Adaptors and Configurators in an unicast or multicast environment.



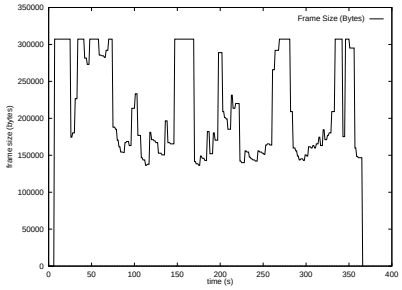
(a) Observed Throughput



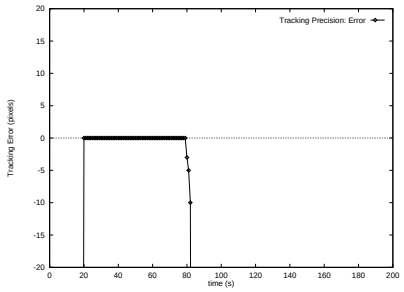
(b) Observed Throughput



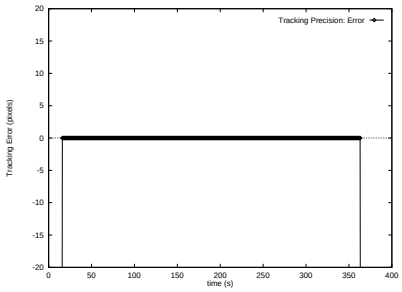
(c) Parameter-tuning Action: Chopped Image Size



(d) Parameter-tuning Action: Chopped Image Size



(e) Tracking Precision



(f) Tracking Precision

Figure 7 (Left): Experiments without adaptation support

Figure 7 (Right): Experiments with adaptation support

Figure 7: Experiments with the Client-Server Based Visual Tracking Application

Appendix A: Internal Mechanisms in the Fuzzy Inference Engine

The fuzzy inference engine operates by using the dual concepts of generalized modus ponens and compositional rule of inference [4].

The concept of generalized modus ponens is derived from the operation of modus ponens in binary logic. Modus

ponens is the operation to draw a conclusion from two premises. Assume that we have the proposition p : "x is A" and the implication if-then rule $p \rightarrow q$: "if x is A then y is B" as true, we can conclude that the proposition q : "y is B" has to be true. In fuzzy logic theory, Generalized modus ponens extends the above operation in the following manner. If we have propositions p : "X is A" and q : "Y is B" where X and Y are linguistic variables and A and B are linguistic values, when both the if-then implication rule $p \rightarrow q$: "if X is A then Y is B" and proposition p^* : "X is A^*" is valid, where A^* is not necessarily the same as A, we can perform the generalized modus ponens and conclude q^* : "Y is B^*". The membership function μ of B^* is calculated by using the $\sup - *$ compositional rule of inference and Larsen's product operation rule:

$$\mu_{B^*}(y) = \sup_x [\mu_{A^*}(x) \star \mu_A(x)\mu_B(y)] \tag{8}$$

where $\star$ is a t-norm operator. An usual selection is the intersection definition of t-norm: $u \star w = \min(u, w)$.

When multiple input linguistic variables exist in the rule, inference can be extended by interpreting the fuzzy set of $A^{(k)}$, which is $\tilde{A}^{(k)}$, as the product of fuzzy sets $A_1^{(k)}, \ldots, A_n^{(k)}$. Its membership function is defined as:

$$\mu_{A_1^{(k)} \times \ldots \times A_n^{(k)}}(x_1, \cdots, x_n) = \mu_{A_1^{(k)}}(x_1) \star \mu_{A_2^{(k)}}(x_2) \star \cdots \star \mu_{A_n^{(k)}}(x_n) \tag{9}$$

where $\star$ is the previously defined t-norm operator and $k = 1, \ldots, m$.

If a rule base contains multiple rules, overall decision of the inference engine is obtained by taking the union of $\tilde{B}^{(k)*}(k = 1, \ldots, m)$, which is the fuzzy sets of linguistic values $B^{(k)*}$ calculated by Equation (8) and (9). The calculation is as follows:

$$\mu_{B^{(1)*} \cup \ldots \cup B^{(m)*}}(y) = \mu_{B^{(1)*}}(y) \otimes \cdots \otimes \mu_{B^{(m)*}}(y) \tag{10}$$

where $\otimes$ represents the s-norm operator for defining disjunctions in approximate reasoning. A usual selection is $u \otimes w = \max(u, w)$.

Appendix B: The Fuzzification Process

A fuzzy inference engine calculates fuzzy sets as results, taking fuzzy sets as inputs. In the above equations, the calculated union of fuzzy sets $\tilde{B}^{(k)*}(k = 1, \ldots, m)$ is the output of the inference engine, while the inference rules and the fuzzy set $\tilde{A}^*$ are the inputs.

However, we do not normally have the fuzzy set $\tilde{A}^*$ in advance, since we normally deal with numerical crisp values. The fuzzification process takes the numerical crisp value x_{in} as input, and generates a fuzzy set $\tilde{A}^*$. If there is no uncertainty in the numerical values, a simple fuzzification process can be:

$$\mu_{A^*}(x) = \begin{cases} 1, & \text{if } x = x_{in} \\ 0, & \text{if } x \neq x_{in} \end{cases} \tag{11}$$

Otherwise, if there is some uncertainty in the numerical value x_{in}, the membership values of the elements of $\tilde{A}^*$ can be selected such that, $\mu_{A^*}(x)$ is taken as 1 if $x = x_{in}$, and $\mu_{A^*}(x)$ decreases linearly from 1 as x moves farther away from x_{in}.

In the former case where no uncertainty is involved, since $\tilde{A}^*$ will contain only a single element with membership value equal to 1, calculation in Equation (8) will become

$$\mu_{B^*}(y) = \mu_A(x_{in})\mu_B(y) \tag{12}$$

In the case of multiple input variables, we substitute Equation (9) in (12) and obtain

$$\mu_{B^*}(y) = \min[\mu_{A_1}(x_{in}), \cdots, \mu_{A_n}(x_{in})]\mu_B(y) \tag{13}$$

to compute the output of one inference rule. Finally, we compute an overall decision by applying Equation (10) to aggregate the calculated $\tilde{B}^{(k)*}, k = 1, \ldots, m$. This shows that the simple fuzzification process shown in Equation (11) simplifies the inference process in the inference engine.

Appendix C: The Defuzzification Process

The *Center of Gravity* method is a frequently used method for the defuzzification process. This method divides the integral of the area under the membership function of the output fuzzy set (Equation 13) into half, and the defuzzified value y_{out} marks the dividing point. Formally in the continuous case, this results in

$$y_{out} = \frac{\int y \mu_{B^*}(y) dy}{\int \mu_{B^*}(y) dy} \tag{14}$$

Acknowledgment

The authors thank graduate students Jun-Hyuk Seo, Kihun Kim, Won Jeon and Seung-won Hwang of University of Illinois at Urbana-Champaign for their useful contributions to the visual tracking project.

References

[1] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A Distributed Real-Time MPEG Video Audio Player. *Proceedings of the 5th International Workshop on Network and Operating System Support of Digital Audio and Video (NOSSDAV'95)*, April 1995.

[2] Z. Chen, S. Tan, R. Campbell, and Y. Li. Real Time Video and Audio in the World Wide Web. *Proceedings of Fourth International World Wide Web Conference*, 1995.

[3] J. DeMeer. On the Specification of End-to-End QoS Control. *Proceedings of 5th International Workshop on Quality of Service '97*, May 1997.

[4] D. Driankov, H. Hellendoorn, and M. Reinfrank. *An Introduction to Fuzzy Control*. Springer-Verlag, 1996.

[5] G. Franklin and J. Powell. *Digital Control of Dynamic Systems*. Addison-Wesley, 1981.

[6] F. Goktas, J. Smith, and R. Bajcsy. Telerobotics over Communication Networks: Control and Networking Issues. *36th IEEE Conference on Decision and Control*, 1997.

[7] A. Hafid and G. Bochmann. Quality of Service Adaptation in Distributed Multimedia Applications. *ACM Springer-Verlag Multimedia Systems Journal*, 6, 1998.

[8] G. Hager and K. Toyama. The XVision System: A General-Purpose Substrate for Portable Real-Time Vision Applications. *Computer Vision and Image Understanding*, 1997.

[9] J. Huang, Y. Wang, and F. Cao. On developing distributed middleware services for QoS- and criticality-based resource negotiation and adaptation. *Journal of Real-Time Systems*, 1998.

[10] D. Hull, A. Shankar, K. Nahrstedt, and J. Liu. An End-to-End QoS Model and Management Architecture. *Proceedings of IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, December 1997.

[11] Object Oriented Concepts Inc. ORBacus for C++ and Java. *ftp://ftp.ooc.com/pub/OB/3.1/OB-3.1b1.pdf*, 1998.

[12] B. Li and K. Nahrstedt. A Control Theoretical Model for Quality of Service Adaptations. *Proceedings of Sixth International Workshop on Quality of Service*, 1998.

[13] C. Parris, G. Ventre, and H. Zhang. Dynamic Management of Guaranteed-Performance Multimedia Connections. *ACM Springer-Verlag Multimedia Systems Journal*, 1994.

[14] A. Pitsillides, Y. Sekercioglu, and G. Ramamurthy. Effective Control of Traffic Flow in ATM Networks Using Fuzzy Explicit Rate Marking (FERM). *IEEE Journal on Selected Areas in Communications*, 1997.

[15] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A Resource Allocation Model for QoS Management. *18th IEEE Real-Time System Symposium*, 1997.

[16] R. Ribler, H. Simitci, and D. Reed. The AutoPilot Performance-Directed Adaptive Control System. *http://www-pablo.cs.uiuc.edu/Publications/publications.htm*, 1997.

[17] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On Adaptive Resource Allocation for Complex Real-Time Applications. *18th IEEE Real-Time System Symposium*, 1997.

[18] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. *Proceedings of the ACM Symposium on Principles of Distributed Computing*, 1996.

[19] B. Stiller, D. Bauer, G. Caronni, C. Class, C. Conrad, B. Plattner, M. Vogt, and M. Waldvogel. Da CaPo++ - Communication Support for Distributed Applications. *TIK Report No. 25, ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report25.ps*, 1997.

[20] N. Yeadon, F. Garcia, A. Campbell, and D. Hutchison. QoS Adaptation and Flow Filtering in ATM Networks. *Proceedings of the Second International Workshop on Multimedia: Advanced Teleservices and High Speed Communication Architectures*, 1994.