A Control-Based Middleware Framework for Quality of Service Adaptations

Baochun Li, Klara Nahrstedt

Abstract— **In heterogeneous environments with performance variations present, multiple applications compete and share a limited amount of system resources, and suffer from variations in resource availability. These complex applications are desired to adapt themselves and to adjust their resource demands dynamically. On one hand, current adaptation mechanisms built within an application cannot preserve global properties such as fairness; on the other hand, adaptive resource management mechanisms built within the operating system are not aware of data semantics in the application.**

In this paper, we present a novel *Middleware Control Framework* to enhance the effectiveness of QoS adaptation decisions by dynamic control and reconfiguration of internal parameters and functionalities of a distributed multimedia application. Our objective is to satisfy both system-wide properties (such as fairness among concurrent applications) and application-specific requirements (such as preserving the critical performance criteria). The framework is modeled by the *Task Control Model* and the *Fuzzy Control Model*, based on rigorous results from the control theory, and verified by the controllability and adaptivity of a distributed visual tracking application. The results show validation of the framework, i.e., critical application quality parameter can be preserved via controlled adaptation.

Keywords— **Application-aware QoS adaptation, Middleware**

I. Introduction

In heterogeneous end-to-end computing environments, it is commonly observed that QoS-aware system components may coexist with QoS-unaware components in the end systems and networks along the end-to-end path. From the application's point of view, if services with statistical or no guarantees exist in the underlying environment, the QoS level that the application demands may not be satisfied continuously. Violations of application QoS requirements may be caused by physical resource limitations such as inherent bandwidth variations and error characteristics in wireless links, or by statistical multiplexing and concurrency introduced by a dynamic number of application tasks sharing the same resource pool in end systems and networks.

Complex distributed applications, residing on top of the above described heterogeneous environment, must be *flexible* and adapt to the QoS variations in their end-to-end execution. They must demonstrate several important properties. First, they need to accept and tolerate resource scarcity to a certain minimum bound, and can improve its performance if given a larger share of resources. Second, when QoS variations occur, they are willing to sacrifice and trade off the quality of less critical parameters for the quality of critical parameters.

An example of flexible applications is the client-server based *distributed visual tracking* application. In this application, the server captures live video images off the camera, and sends them over the network to the client. The client accepts user input (visual specification of desired tracked objects), receives the video, executes multiple computationally intensive tracking algorithms (referred to as *trackers*) on corresponding moving objects in the video, and displays the video with illustrations of tracking results. The complexity of the application is caused by the mapping between the quality of critical parameters, such as the *tracking precision*[1], and resource availability in the system. As long as the tracking precision can be preserved, the application is very flexible with respect to other quality parameters, and can be dynamically tuned or reconfigured in order to adapt to variations in both CPU and bandwidth availability. The structure of the application is shown in Figure 1.

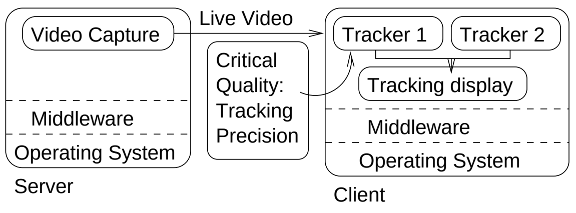


Fig. 1. Client-Server Based Distributed Visual Tracking

If we examine the adaptation efforts throughout various levels in a distributed system, we note that there are two distinctive levels where adaptation may take place: the system level (e.g. operating systems and network protocols) and the application level. Adaptation practices in these two categories have different objectives. In the system level, the objectives of adaptation are set at a much lower level, for example packets or cells in network protocols [1], and time slices in adaptive soft real-time scheduling algorithms for the CPU resource [2]. In addition, the emphasis is focused on global properties such as fairness or overall resource utilization. On the other hand, adaptations in the application level are more focused on higher level application-specific semantics, such as frame rate in video streaming [3] or precision in visual tracking, and optimized for the favor of its

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

The authors are with Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL. E-mail: {b-li,klara}@cs.uiuc.edu.

[1]The *tracking precision* is defined as the distance between the center of tracked region and center of the object. We wish to keep this distance stable and as small as possible.

own performance, such as minimum degradation of perceptual quality, or stable tracking precision.

In this work, in order to balance the adaptation objectives of both system and application levels, we focus on the study of *application-aware QoS adaptations* [4][5]. We propose to introduce a *middleware control framework* to support application-aware adaptations, which is aware of both system-wide requirements such as stability and fairness, and application-specific choices for adaptive mechanisms.

In the following sections, we first present a general overview of the middleware control framework in Section II and then divide the discussion into two parts. The first part presents the theoretical models, i.e., the Task Control and Fuzzy Control Model, of the framework in Section III and IV. The second part discusses the design and implementation issues of the framework in Section V, using the distributed visual tracking application as an example. Section VI shows experimental results with the distributed visual tracking application. Section VII discusses related work, and Section VIII concludes the paper.

II. Overview of Middleware Control Framework

The services provided by the *middleware control framework* have three major objectives. First, they serve as a 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 strengthen the adaptation awareness and effectiveness within 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 on-the-fly dynamics in the heterogeneous environment, so that informed decisions can be made to control the applications. Without the introduction of middleware control framework, applications are required to make decisions by themselves, thus required to implement observation methods for respective system resources and proper adaptation policies within the application. This increases the burden of application development, and proprietary implementations that are tightly bound to a specific application cannot be shared with other applications, even with similar semantics and behavior.

The design of our middleware control framework uses a hybrid approach based on two major models: the *Task Control Model* and the *Fuzzy Control Model*. The design objective of the Task Control Model is to make appropriate adaptation decisions to ensure system-wide properties are satisfied, such as fairness among concurrent applications or adaptation stability. The objective of the Fuzzy Control Model, on the other hand, is to map these system-level adaptation decisions to application-specific tuning or reconfiguring adaptation choices, so that the quality of critical application parameters (e.g. tracking precision) can be preserved.

Within the middleware control framework, the Task Control Model and Fuzzy Control Model are represented in two respective components: the *Adaptor* and the *Configurator*. The *Adaptor* consists of the *Adaptation Task* and *Observation Task*, which include the characteristics and algorithm of the Task Control Model. Each Adaptor corresponds to a specific type of resource, most notably CPU and network bandwidth. All concurrent applications are controlled by one Adaptor for each type of system resource, in order to maintain fairness and stability. On the other hand, the *Configurator* includes the Fuzzy Control Model, and maps adaptation decisions made by the Adaptor to application-specific parameter-tuning and reconfiguring adaptation choices within the application. Thus, each application needs to have a corresponding Configurator. The relationships among these components are shown in Figure 2.

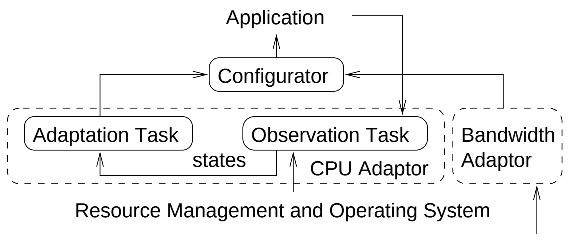


Fig. 2. Components in the Middleware Control Framework

In our case study, a distributed visual tracking application is deployed under the control of the middleware framework. We will use running examples derived from this application to present the theoretical results in the Task Control and Fuzzy Control Models, discussed in Sections III and IV, respectively.

III. Task Control Model

A. Motivation

One of the major features of the *Task Control Model* that differs from previous adaptation schemes is that it focuses on *actively controlling* the adaptation behavior of applications, rather than simply providing hints to applications about current system states via upcalls, so that applications can adapt by itself. The justifications of adopting an *active control* approach are derived by the following observation of a typical control system in control theory [6].

If we examine the essential characteristics of the adaptation process, it corresponds naturally to a typical control system. In a control system, there is a target system to be controlled. The internal states within the target are determined by a *controller* according to a *control algorithm.* The output of the control algorithm is determined based on the states observed in the target system, by comparing them to the desired values referred to as the *reference* or *setpoint.* The *observer* is responsible to observe the states in the target. The objective of the control system is to steer the target system so that it is not affected by disturbances. In comparison, the adaptation process in an end system follows an identical model.

Hence, we derive the following insights from this analogy to the control system. The previous schemes, where applications adapt based on upcalls or notifications from the system, actually integrate the controller into the application, while leaving the system to serve as observers. We propose to detach the controller from the application, so that all concurrent applications can benefit from a *unified controller* design. The role of this controller is implemented by the *Adaptor* in the middleware control framework. This design strategy forms the basis of the Task Control Model.

The above design strategy adopted by the Task Control Model leads to two major advantages. First, because of the *unified controller* design, it is straightforward to utilize the control theory in order to formally derive and prove convergence, fairness at equilibrium and stability properties. Second, it cleanly separates highly application-specific adaptation mechanisms, such as parameter tuning, from the control algorithms used to determine adaptation timings and scale. It makes it possible to flexibly associate control signals produced by the control algorithms with various application-specific adaptive mechanisms, such as buffer smoothing, media scaling and filtering, and with functional reconfigurations as well.

B. Task Flow Model

In order to design control algorithms using the control theory, we need a strict mapping between models used in control systems and our design of active adaptation control. For this purpose, we consider each application as an ensemble of functional components, which we refer to as *tasks*. Tasks are execution units that consume system resources and perform certain actions to deliver a result to other tasks or the end user.

With the above definition of *tasks*, we utilize the Task Flow Model [7] to address the modeling of active adaptation control. In the Task Flow Model, a Task Flow Graph, which is a directed acyclic graph consisting of multiple tasks, is formed for each application. A directed edge from task T_i to task T_j indicates that task T_j uses the output produced by task T_i. Figure 3 illustrates the Task Flow Graph for the distributed visual tracking application.

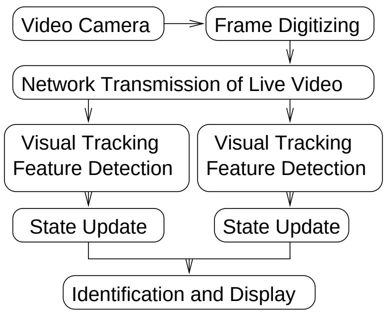

Video Camera
Frame Digitizing
Network Transmission of Live Video
Visual Tracking Feature Detection
Visual Tracking Feature Detection
State Update
State Update
Identification and Display

Fig. 3. Task Flow Graph for Distributed Visual Tracking

C. Task Control Model

Based on the Task Flow Model, the *Task Control Model* concentrates on a single task in the Task Flow Graph, referred to as the *Target Task*, which is the task to be controlled. Based on derived analogy to control systems, we introduce the following tasks that are deployed in the Task Control Model: (1) *Adaptation Task*. It serves as the *controller* and implements the control algorithm. The output of the Adaptation Task is a series of control signals that are used to control the application, via application-specific adaptation choices. (2) *Observation Task*. It observes the *task states* in the Target Task, and feeds them back to the Adaptation Task. *Task states* are defined as the internal parameters within the Target Task that represent its dynamics of resource consumption. For example, in the distributed visual tracking application, *frame rate* can be one of the task states related to network bandwidth, and *tracking frequency*[2] can be one of the task states related to CPU. Figure 4 illustrates the Task Control Model.

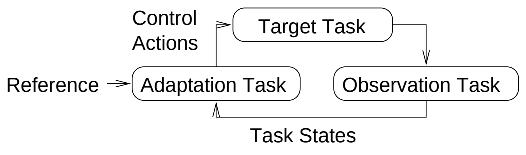


Fig. 4. The Task Control Model

The ideal objective of the Task Control Model is to achieve the following properties: (1) The Target Task is controlled to maintain stable quality of its critical QoS parameters, regardless of variations in resource availability. For example, *tracking precision* should be preserved at a stable level in distributed visual tracking. (2) The adaptation process is carried out in a timely fashion, when detecting and responding to changes in resource availability. This property is referred to as *agility* of adaptation in [5]. We discuss adaptation agility in Section III-D.3. (3) In order to accommodate concurrency of resource accesses among multiple applications, fairness of adaptation needs to be balanced among all competing tasks. This property can be held if the Adaptation and Observation Task have *complete control* and *complete knowledge* about *all* concurrent applications sharing the resource, which applies to any end system resources, such as CPU, memory, storage space and network host interface.

C.1 Generic Model of Target Task

In order to utilize the control theory to model the adaptation process in the Task Control Model, we need to derive a suitable model for the Target Task. In the most generic fashion, if we use the vector $\mathbf{x}$ for the vector of task states, the vector $\mathbf{u}$ for the vector of controllable input parameters, the vector $\mathbf{z}$ for the vector of observed output parameters of the task, the vector $\mathbf{w}$ for the uncontrollable variations in the task, and the vector $\mathbf{v}$ for the observation errors, we can model the *Target Task* with the following equations:

$$\frac{d\mathbf{x}(t)}{dt} \equiv \dot{\mathbf{x}}(t) = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t), t] \qquad (1)$$

[2] *Tracking frequency* is defined as the number of iterations that one or more trackers can execute each second. Trackers are executed in a round-robin fashion in each iteration. Since trackers are computationally intensive, the *tracking frequency* is highly dependent on CPU resource availability.

$$\mathbf{z}(t) = \mathbf{h}[\mathbf{x}(t), \mathbf{v}(t), t] \quad (2)$$

With the above definition, the task is said to be at *equilibrium* when:

$$\dot{\mathbf{x}}(t) = 0 = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t), t] \quad (3)$$

An equilibrium is *stable* if small disturbances do not cause the state to diverge and oscillate. Otherwise, it is an unstable equilibrium. From a practical point of view, this formal definition of *adaptation stability* is identical to the definitions in previous work [8] [9], where *stability* is defined as the ability of the system to steer the target system back to equilibrium state after disturbances have occurred.

The above stated definitions are generic and may be continuous in time, non-linear and time-varying. In this paper we study a subset, namely, the tasks that can be approximated piecewise without loss of accuracy by discrete and linear equations of the following form:

$$\mathbf{x}(k) = \mathbf{\Phi}\mathbf{x}(k-1) + \mathbf{\Gamma}\mathbf{u}(k-1) + \mathbf{w}(k-1) \quad (4)$$

$$\mathbf{y}(k) = \mathbf{H}\mathbf{x}(k) \quad (5)$$

$$\mathbf{z}(k) = \mathbf{y}(k) + \mathbf{v}(k) \quad (6)$$

where $k = 1$ to k_{max}, and $\mathbf{\Phi}$, $\mathbf{\Gamma}$, and $\mathbf{H}$ are known transition matrices without an error. The above generic linear model is frequently used in the state-space approach in control systems.

C.2 Concrete Model of Target Task

In order to derive properties in the control algorithm, we need to develop a concrete model for the Target Task, based on the generic linear model presented. We thus consider the following scenario.

Let us assume multiple tasks competing for a shared resource pool with the capacity C_{max}. Each task makes *requests* 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 in the outstanding status until it is granted. The system is granting c requests within a given time interval for multiple running tasks.

The mapping between the abstract notation *resource requests* and the actual services that process the resource requests varies among different types of system resources. For *temporal* resources, such as central processor capability and transmission throughput, where the resources are shared in a temporal fashion, outstanding resource requests may be mapped to the waiting queue, and granted requests may be mapped to allocated temporal resources, such as bandwidth. For *spatial* resources, such as volatile or non-volatile storage capacity, outstanding requests may be mapped to the actively used and occupied capacity, such as temporal buffers in caches and real memory (swapped in pages), and granted requests may be mapped to the reclaimed capacity by the system due to inactivity, such as swapped out pages. The model presented in this section applies to both cases.

To be more exact, we take CPU and network bandwidth as two examples of all resource types. For network bandwidth resource, for example in the client-server based distributed visual tracking application, outstanding resource requests can be interpreted as those data units[3] that are in transit between the server and the client. Granted resource requests, in this example, can be interpreted as the data units that have been received at the server.

For CPU resource, since allocation of CPU is on a time-sharing basis[4], the interpretation of the above abstract notions is more complex. In order to do this, we can first construct an ideal case where full CPU capacity are available to the application, i.e., the application is running at the top possible speed. We can then refer to the resource requests within the application in the ideal case as desired reference value. In actual cases, we interpret *granted resource requests* as the actual requests in the application, and *outstanding resource requests* as the difference between the desired number of requests and actual granted number of requests. For example, in the distributed visual tracking application, we assume that the *tracking frequency* is 30 iterations per second when the system is running at full CPU capacity. In the actual cases, if the observed tracking frequency is 10 iterations per second, the outstanding resource request rate is 20.

We further categorize the resource types into *completely controllable* resources and *partially controllable* resources. All concurrent tasks competing for *completely controllable* resources, such as CPU, are observable and their behavior controllable. For these tasks, fairness of resource usage can be enforced by appropriate adaptation control. On the other hand, for *partially controllable* resources, such as network bandwidth, not all concurrent tasks sharing the same resource can be observed and controlled in one end system. Only transient properties of adaptation behavior can be enforced, such as stability and agility. We discuss completely controllable resources in this subsection (III-C.2) below, and postpone separate discussions on bandwidth-related adaptation to Section III-E.

Figure 5 shows the Target Task that accesses the shared resource pool. Resource management and scheduling mechanisms grant resources to concurrent tasks, and requests that are not granted are treated as outstanding. The Adaptation Task controls the adaptation process of the Target Task by controlling its *new request rate* of resources, so that it does not exceed its fair share. The control is enforced within the application by parameter-tuning or reconfiguring options.

In the Task Control Model, we define the following variables corresponding to a *Target Task* T_i:

[3] At the system level, data units may be cells or packets. However, at the application level, the actual interpretation of *data units* depend on the application-specific semantics. For example, in video streaming applications a data unit may be interpreted as a *frame*.

[4] An instruction at the assembly level of the application is either issued to the CPU or blocked for its time slice by the operating system scheduler.

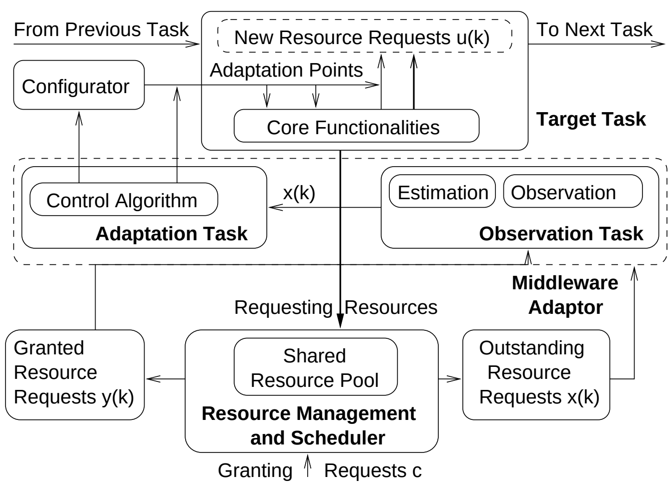


Fig. 5. A Concrete Model for the Target Task

1. c is the total number of granted requests in $[k, k+1]$ for all Target Tasks in case of completely controllable resources.
2. t_c is a *constant sampling time interval*, which is the time elapsed in interval $[k, k+1]$, k being time instants.
3. $u_i(k)$ is the *number of resource requests* during $[k, k+1]$ allowed by the *Adaptation Task* of T_i, which is referred to as the *adapted request rate.*
4. $r_i(k)$ is the original desired *number of resource requests* made by T_i during $[k, k+1]$. If $r_i(k) < u_i(k)$, i.e., the Target Task requests less resources than the limit for Adaptation Task allows, then $u_i(k) = r_i(k)$. If $r_i(k) > u_i(k)$, then the application adapts. Note that this condition allows some applications to be greedy. $r_i(k)$ is the inherent decision of the application, while $u_i(k)$ is determined by the Adaptation Task.
5. $x(k)$ is the *total number* of *outstanding resource requests* made by *all tasks* at time k;
6. $M(k)$ is the total number of active Target Tasks competing for resources in the system;
7. $A(k)$ is the set of *adapted* Target Tasks at time k who satisfy $r_i(k) > u_i(k)$, $N(k)$ is the set of Target Tasks that do not adapt and satisfy $r_i(k) = u_i(k)$.
8. $l(k)$ is the number of tasks in $A(k)$ ($l(k) = |A(k)|$), $M(k) - l(k)$ is the number of tasks in $N(k)$ ($|N(k)| = M(k) - l(k)$). We assume that both $M(k)$ and $l(k)$ stay constant within one time interval $[k, k+1]$.
9. w_i is the static weight of T_i showing its importance compared to other Target Tasks.

Using the above notations, an approximation of the derivative of outstanding resource requests can be described as follows:

$$\frac{x(k) - x(k-1)}{k - (k-1)} = x(k) - x(k-1) = \sum_{i=0}^{M(k-1)} u_i(k-1) - c \tag{7}$$

The Difference Equation (7) depicts the internal dynamics of the adaptive system.

The objective of the control is to maintain the number of total outstanding requests x to stay around a specific *reference* value $x^c(k)$ (conceptually similar to a specified burst length in leaky bucket). Under the assumption that the adaptive system behaves according to Equation (7), we can derive a control algorithm in the *Adaptation Task* for Target Task T_i to calculate $u_i(k)$ values, which will lead to the desired values for x. For example, if a standard proportional-integral-derivative (PID) control [6][5] is engaged, then $u_i(k)$ obeys the equation

$$u_i(k) = u_i(k-1) + \alpha[x^c(k) - x(k)] + \\ \beta\{[x^c(k) - x(k)] - [x^c(k-1) - x(k-1)]\} \tag{8}$$

where α and β are configurable scaling factors. The reason for adopting PID as our control algorithm is as follows. (1) Shown in control theory, this algorithm belongs to simple linear algorithms and shows good performance with respect to eliminating steady-state errors, rejecting disturbances, damping transient response and stabilizing adaptation behavior. It is the most effective and widely used control algorithm. (2) Shown in later sections, we are able to prove stability conditions and fairness properties at equilibrium. (3) It is simple to be implemented with little overhead related to computational complexity.

D. Control Theoretical Analysis of the Adaptation Task

This section continues the discussion of the Task Control Model with a rigorous analysis of various properties of the adaptation process. The Task Control Model is characterized by the model of *Target Task* expressed in Equation (7), as well as the control algorithm in Adaptation Task expressed in Equation (8).

The PID control algorithm presented in Equation (8) can be rewritten as follows:

$$u_i(k) = \Psi_{r_i(k)}\ \{\ u_i(k-1) + \alpha(x^c(k) - x(k)) \\ +\beta[(x^c(k) - x(k)) - (x^c(k-1) - x(k-1))]\ \} \tag{9}$$

Where $\Psi_b(a)$ is defined as:

$$\Psi_b(a) = \begin{cases} 0 & \text{if } a < 0, \\ b & \text{if } a > b, \\ a & \text{otherwise.} \end{cases} \tag{10}$$

In addition, since at time k we assume that, among all Target Tasks, $l(k)$ tasks are adapted by their respective adaptation tasks (where $u_i(k) < r_i(k)$), and $M(k) - l(k)$ tasks are not affected (where $u_i(k) = r_i(k)$), we conclude that the total number of outstanding resource requests in the system shows the following dynamic property (an extension to Equation (7)):

$$x(k) = \Psi_{C_{max}}\{x(k-1) + \sum_{T_i \in A(k-1)} u_i(k-1) + \\ \sum_{T_i \in N(k-1)} r_i(k-1) - c\} \tag{11}$$

[5]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.

$$= \Psi_{C_{max}}\{x(k-1)+l(k-1)\bar{u}(k-1) + R(k-1)-c\}, \text{ where} \quad (12)$$

$$R(k) = \sum_{T_i \in N(k)} r_i(k), \text{ and} \quad (13)$$

$$u_i(k) = \gamma_i(k)\, l(k)\, \bar{u}(k) \quad (14)$$

where $\bar{u}(k)$ is the average rate for all $u_i(k)$ that satisfy $u_i(k) < r_i(k)$. In this equation, $\gamma_i(k)$ is the *dynamic weight* of task T_i which indicates priority for resource requests, and satisfies $\sum_{T_i \in A(k)} \gamma_i(k) = 1$. The dynamic weight of T_i can be derived from the *static weight* w_i of T_i, with the following calculation:

$$\gamma_i(k) = \frac{w_i}{\sum_{T_j \in A(k)} w_j} \quad (15)$$

Intuitively, if at time k all Target Tasks adapt, then $\gamma_i(k) = w_i$. Otherwise, $\gamma_i(k)$ is the importance compared to other adaptive Target Tasks.

Combining Equation (9) and Equation (12), we obtain the complete characterization of the adaptation system.

D.1 Equilibrium Analysis

Now that we have established the control algorithm in the Adaptation Task, we continue to analyze the equilibrium properties of the system. The ideal case is that $x(k)$ converges towards reference $x^c(k)$, while $u_i(k)$ converges to the fair share of each competing task. Let us assume that for a specific period of time $[k_1, k_2]$, $x^c(k)$, $l(k)$, $M(k)$ and $R(k)$ are all stable and stay at constants x^c_s, l_s, M_s and R_s, respectively. Then we show the following properties:

Theorem 1: Within $[k_1, k_2]$, the number of outstanding resource requests x in the system, established by Equation (9) and (12), will converge to an equilibrium value which equals to the reference value x^c_s. In addition, the system also fairly shares resources among competing tasks based on their static weights, according to the *weighted max-min* fairness property.

Proof: The proof is presented in Appendix A. ■

We note that the fairness property can only be preserved when task states for all applications competing for shared resources can be observed, i.e., for completely controllable resources, such as CPU. It normally does not hold in the case of partially controllable resources, such as network bandwidth resources, where resources are shared with connections from foreign end systems.

D.2 Stability Analysis

Similar to the definitions in previous work [8][9], the concept of *adaptation stability* is related to two categories of the system dynamics. (1) *Statistical multiplexing.* In an environment of multiple Target Tasks simultaneously sharing the limited availability of resources, when the number of active Target Tasks is fixed, system resources allocated to each Target Task should settle down to an equilibrium value in a definite period of time. This also implies that, if a new task becomes active, existing active tasks will adjust their resource usage so that after a brief transient period, the system settles down to a new equilibrium. (2) *Disturbances.* Stability also implies that with respect to variations in resource availability due to unpredictable and physical causes, such as a volatile wireless connection, adaptation activities do not suffer from oscillations. Oscillations are undesirable because they cause both fluctuations in user-perceptible qualities, and an excessive amount of adaptation attempts that may occupy so much resources that the system is overloaded.

In order to converge to the equilibrium of the system regardless of statistical multiplexing and disturbances, we need to prove that the system is stable. Due to the global nonlinear nature of the system given by upper and lower bounds in Equation (9) and (12), we are unable to derive a global and absolute stability condition. However, formal conditions for asymptotic stability can be addressed analytically around a local neighborhood, in which the system is linear. We present the following theorem related to local asymptotic stability conditions.

Theorem 2: The adaptation system established by Equation (9) and (12) is asymptotically stable for task T_i around a local neighborhood, under the condition that $\alpha > 0$, $\beta > 0$, and $\alpha + 2\beta < 4\gamma_i$.

Proof: The proof is presented in Appendix B. ■

We can derive from Theorem 2 that the asymptotic stability for task T_i is determined by an appropriate choice of α and β. It then follows that in order to guarantee that the entire system is stable, we need to choose α and β so that for any task T_i with any static weight values w_i, stability is ensured.

Corollary: There exist appropriate values of parameters α and β so that all tasks in the system are stable, for any pre-determined static weight w_i for task T_i.

Proof: Assume w_{min} is the minimum value among all pre-determined w_i. α and β can be chosen to satisfy

$$\alpha > 0,\ \beta > 0, \ \text{ and}$$

$$\alpha + 2\beta < 4\,\frac{w_{min}}{\sum_{\forall i} w_i} < 4\,\frac{w_i}{\sum_{T_j \in A(k)} w_j},\ \forall i,\ k \quad (16)$$

It follows from Theorem 2 that if these conditions hold, the system will be stable for any task T_i with a static weight w_i. ■

D.3 Adaptation Agility

In addition to stability requirements, it is also desired that the system responds quickly to variations in resource availability. *Adaptation agility* [5] is defined as the speed and accuracy of an adaptive system with respect to detecting and responding to changes in resource availability and application QoS requirements. In short, it is the *responsiveness* or *sensitivity* of an adaptive system to dynamic variations. This notion is also similar to the notion of *sensitivity* introduced in [9].

In the Task Control Model, adaptation agility is determined by configurable parameters in the control algorithm. In the case of a PID control algorithm in Equation (9), α

and β are configurable as long as the stability conditions in Theorem 2 hold. The actual configuration is tailored to the agility needs of the system. Some applications, for example the distributed visual tracking, prefer to adapt with a high agility level to variations in resource availability, since the critical parameter, *tracking precision*, is sensitive to even minor perturbations.

The selection strategy of configurable parameters α and β in the PID control algorithm is as follows. First, the combination of α and β settings should satisfy all stability requirements established in Theorem 2 and its corollary. Second, α and β correspond to the *integral* and *derivative* settings of the PID control algorithm, respectively. It is known from the control theory that the integral control increases the speed of transient response. In other words, if α is tuned to be lower, transient response is faster. On the other hand, derivative control improves stability and increases damping. In other words, if β is tuned to be lower, the damping factor is higher, and the system converges slower but is more stable.

Two illustrations are given in Figure 6(a) and 6(b) to show the effects of different configurations on the system. The changes of $l(k)$, $R(k)$ and $x^c(k)$ are given in Table I:

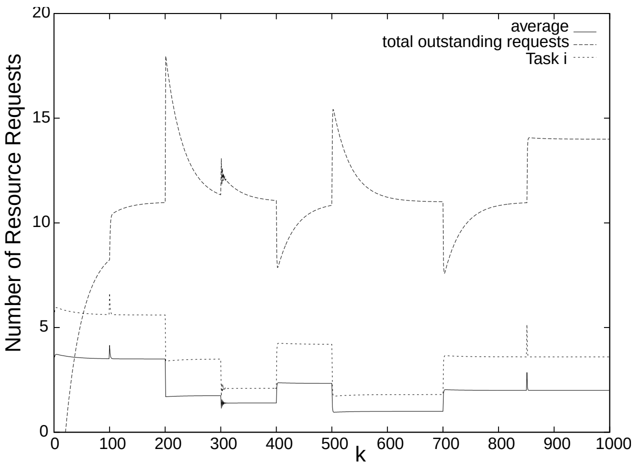
k	*0*	*100*	*200*	*300*	*400*	*500*	*700*	*850*
l	2	2	4	5	3	3	3	3
R	3	3	3	3	3	7	4	4
x^c	9	11	11	11	11	11	11	14

TABLE I
THE VALUES OF $l(k)$, $R(k)$ AND $x^c(k)$ USED IN FIGURE 6

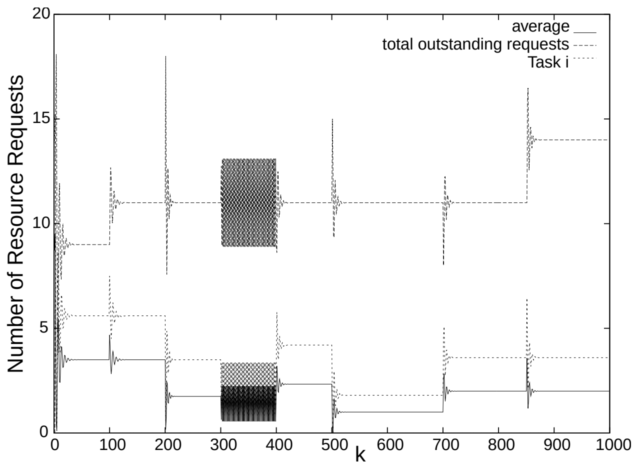
It is shown that different α and β settings have diversely different transient responses. In Figure 6(a), α is low and β is high, which reflect the requirements that the agility should be low and the system should remain stable. Figure 6(b), on the contrary, shows the results when agility is set to be extremely high, with α tuned to be high and β tuned to be low. For example, in the interval $[300, 400]$ (Figure 6(b)), oscillation occurs with the relative weight $\gamma_i = 0.3$ for T_i, which satisfies $\alpha + 2\beta = 4\gamma_i$, and violates the stability conditions derived from Theorem 2. Overall, Figure 6 shows that, by tuning integral and derivative settings α and β in the PID control algorithm, adaptation agility is highly configurable according to the agility needs of the system.

E. Extensions of Task Control Model to Partially Controllable Resources

For completely controllable resources such as CPU, a PID control algorithm was given in the previous subsection, and a weighted max-min fairness property was proved. A prerequisite for the fairness property to hold is that the resource is completely controllable, i.e., that the Observation Task has the ability to observe complete states for all competing tasks, and that the Adaptation Task can completely control competing tasks as well.



(a) $\alpha = 0.015,\ \beta = 0.5$



(b) $\alpha = 0.7,\ \beta = 0.25$

Fig. 6. Configurable Agility and Dynamic Responses

However, this is normally not the case for partially controllable resources. An example of such resource is the network bandwidth in a distributed environment, when observing task states within the Transmission Task. Note that the Transmission Task is a distributed task considering of sending and receiving subtasks, which reside at the sender and receiver end points, respectively. Since the Observation Tasks reside as middleware components in the end system, it has no ability to obtain states corresponding to all other connections sharing the network. In such cases, the only observable states are the parameters used or allocated by the Transmission Task itself. Therefore, while the control algorithm still adapts to variations in resource availability and shows stability and convergence properties, it lacks crucial observations to guarantee any global fairness properties.

In distributed applications, we focus on the *Transmission Task* as the Target Task T_i that consumes bandwidth. Rather than using the total number of outstanding requests

$x(k)$ to model the Target Task, we use $x_i(k)$, which is the number of outstanding requests made by T_i alone. $x_i(k)$ can be interpreted as the number of data units *in flight* in T_i before reaching destination. We assume $u_i(k)$ is the new resource requests by T_i, which is equivalent to the actual number of data units sent by T_i. We also assume that $c_i(k)$ is the number of granted requests in $[k, k+1]$ to T_i, which is the number of data units received by the destination. The model for the transmission task is:

$$x_i(k) = x_i(k-1) + u_i(k-1) - c_i(k-1) \tag{17}$$

Similarly, the PID control algorithm adopted in the Adaptation Task may be modified as follows:

$$u_i(k) = u_i(k-1) + \alpha[x_i^c(k) - x_i(k)] + \beta\{[x_i^c(k) - x_i(k)] - [x_i^c(k-1) - x_i(k-1)]\} \tag{18}$$

where x_i^c is the reference value expected at equilibrium. The stability and convergence proofs still hold as in the previous section, under the assumption that the sender receives the feedback about $x_i(k)$ from the receiver. This means that the end-to-end delay between sender and receiver must be very short, which can be true in LAN environments.

IV. Fuzzy Control Model

A. Overview

In the Task Control Model, dynamic properties of the adaptation process are addressed at the end system, such as stability guarantees, adaptation agility and equilibrium fairness. However, the overall application behavior is nonlinear and it may be possible that some desired QoS parameters cannot be maintained by simple parameter-tuning options. This section introduces the *Fuzzy Control Model*, which focuses on application-specific adaptation choices, with enhanced parameter-tuning possibilities or reconfigurations. The model utilizes results from fuzzy logic and the fuzzy control theory [10].

In the design of the middleware control framework, the Fuzzy Control Model is implemented in the application-specific Configurators. The relationships between the Task Control and Fuzzy Control Models are as follows. (1) The Fuzzy Control Model implements a *mapping process*. It maps the output of the Adaptation Task $u_i(k)$ to application-specific parameter-tuning adaptation choices. (2) The Fuzzy Control Model *extends* the *adaptation process* that is implemented by the Adaptation Task. When parameter-tuning adaptations are not effective to meet the resource availability levels, reconfiguration choices are activated. The timing of these activations is determined by a set of *rules*, which represent a nonlinear control surface. (3) Combining the Task Control and the Fuzzy Control Model, a *hybrid* adaptation scheme is formed, offering an adaptation strategy that approximates nonlinear behavior by using piecewise linear control at normal times and using fuzzy control at transition points. that are both piecewise linear and nonlinear in extreme cases. The Task Control Model provides a basic linear model for the adaptation process, while the Fuzzy Control Model extends the adaptation process to support nonlinear adaptation choices, as well as to provide an application-specific *mapping process* that maps to parameter-tuning adaptation choices. Figure 7 shows this role of the Configurator.

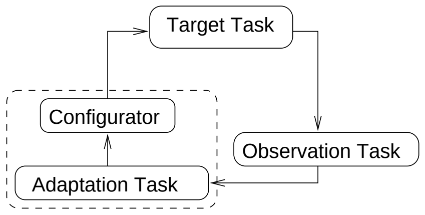


Fig. 7. The Role of Configurator in the Task Control Model

The major advantages of this model are the following. (1) A fuzzy logic approach is used in the design of the mapping and adaptation process, so that application-specific adaptation choices can be expressed explicitly with a *rule base* and *member functions* for each linguistic value. The *rule base* provides *linguistic rules* that the mapping process is based on. (2) With respect to the nonlinear adaptation process based on reconfiguration choices, assuming that multiple discrete reconfiguring options exist in a complex application, the controllable regions are in most cases discrete and non-linear in nature. The Fuzzy Control Model is most appropriate when dealing with nonlinearities of discrete adaptation choices, since the mapping process from inputs and outputs is expressed by using a number of *linguistic rules* stored in the *rule base*. (3) The rules that guide the mapping and nonlinear adaptation process are highly configurable, and can be intuitively specified to satisfy the needs of the application. The model offers a flexible design suitable for new applications, without loss of generality and configurability.

B. Configurator Design: The Fuzzy Control Model

The architecture of the Fuzzy Control Model, shown in Figure 8, includes five components built within the Configurator. The *fuzzy inference engine* implements the fuzzy logic based mapping and nonlinear adaptation process. The adaptation process is based on a set of *linguistic rules* defined in the application-specific *rule base*, as well as on a set of *membership functions* for linguistic values. The fuzzy inference engine takes fuzzy sets as input, and produces output in the form of fuzzy sets. Hence, the architecture also includes the *input normalizer* and *fuzzifier* to prepare input fuzzy sets for the fuzzy inference engine, as well as the *defuzzifier* to convert output fuzzy sets to actual adaptation choices in the application.

The rule base consists of a set of linguistic rules, which are expressed using *linguistic values* and *linguistic variables*. Each linguistic variable can be assigned multiple possible linguistic values. All linguistic values used in the rule base should use words of a natural or synthetic language, for example `moderate` or `below_average` for the linguistic variable `cpu_demand`. These linguistic values are

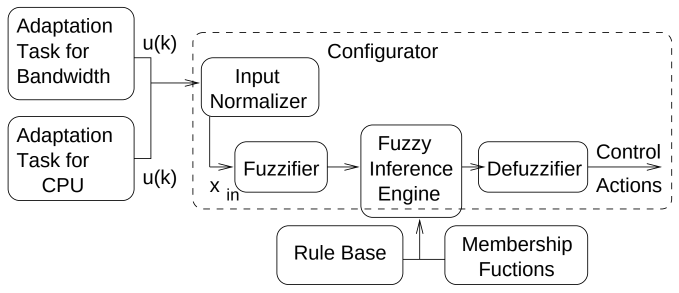


Fig. 8. The Architecture of the Fuzzy Control Model

modeled by fuzzy sets, which are unambiguously expressed by a *membership function*. The linguistic rules and membership functions for all linguistic values in these rules are highly configurable according to application-specific adaptation choices and requirements.

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. The first phase of design generates a set of conditional statements in the form of if-then rules. The generic form is:

$$\begin{array}{ll} R^{(1)}: & \text{if } X_1 \text{ is } A_1^{(1)} \text{ and } \ldots \text{ and } X_n \text{ is } A_n^{(1)} \text{ then } Y \text{ is } B^{(1)} \\ \cdots & \cdots \\ R^{(m)}: & \text{if } X_1 \text{ is } A_1^{(m)} \text{ and } \ldots \text{ and } X_n \text{ is } A_n^{(m)} \text{ then } Y \text{ is } B^{(m)} \end{array} \quad (19)$$

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 the membership functions of their respective fuzzy sets, expressed by $\mu_{A_l^{(k)}}(x)$ 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 the mapping process from the fuzzy set input to fuzzy set output. After the defuzzification process, the fuzzy set output directly corresponds to a particular adaptation choice within the application. For completeness of this paper, the internal mechanisms implemented in the fuzzy inference engine is summarized in Appendix C, the fuzzification process is discussed in Appendix D, and the defuzzification process is presented in Appendix E.

An example of using the Fuzzy Control Model, using the distributed visual tracking application is presented in Section V.

V. Design of Middleware Control Framework

In this section, we present our architectural design of the Middleware Control Framework, with the distributed visual tracking application as a running example.

A. The Architecture

The Adaptors and Configurators in the middleware control framework are implemented as middleware components. There are three unique characteristics and advantages in our design, differing from previous middleware solutions, as follows.

(1) The Middleware Control Framework *enhances* existing service enabling platforms in the middleware level. It is designed to utilize services provided by such platforms, not as replacements. This design principle features a high level of flexibility when applying the framework to new environments and applications, since it can be deployed on top of other service enabling platforms of choice. For example, Real-time CORBA [11] can be *enhanced* by the Middleware Control Framework to provide better adaptation control mechanisms, while still retaining its own features related to real-time guarantees.

(2) The Middleware Control Framework is *active*. Rather than passively providing application-transparent adaptive services, the framework actively controls the adaptation behavior of the application. Since our components, including Adaptors and Configurators, may be implemented in a diversely different environment or programming language, the interactions among components of the active adaptation are made possible by service enabling platforms, such as CORBA.

(3) Since our proposed design interacts with applications only through services provided by existing service enabling platforms, there are no limitations on the behavior of the application. For instance, in the example of CORBA, the applications do not have to communicate with remote components via CORBA services, they have the freedom to choose between sockets, RPC calls or other communication means. The only requirement for these applications is that they export application-specific adaptation choices to the middleware components, specified by CORBA interface specification. Figure 9 shows the architecture of the framework.

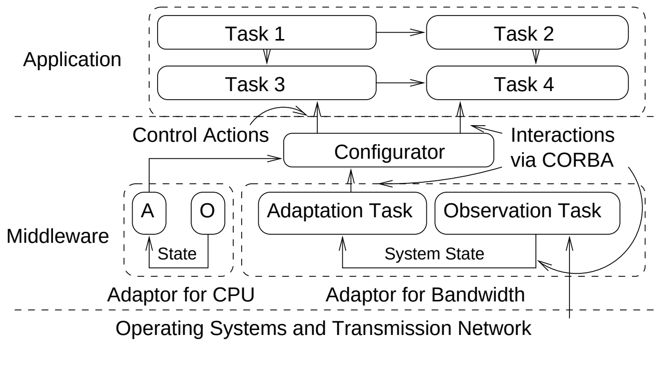


Fig. 9. The Architecture of the Middleware Control Framework

Components in the Middleware Control Framework are implemented locally or in a distributed environment, in order to perform adaptation control of an application, such as distributed visual tracking. The scenario that the Middleware Control Framework is deployed in a client-server based environment is illustrated in Figure 10.

B. Distributed Visual Tracking: An Example

In order to illustrate detailed design on how the Middleware Control Framework controls an application, we use

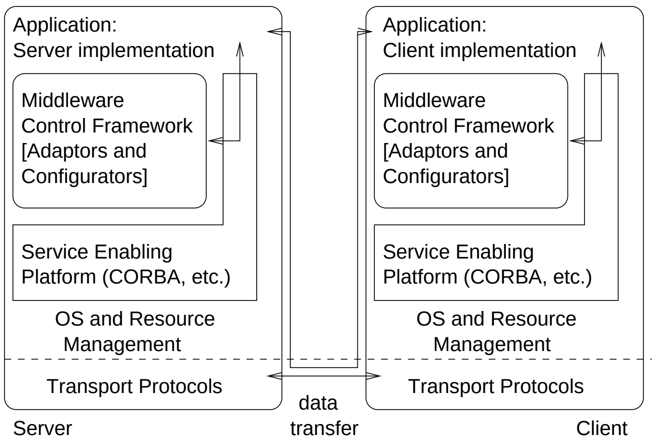


Fig. 10. Middleware Control Framework in a Distributed Environment

the distributed visual tracking application as an example.

B.1 Adaptation Choices

As previously noted, the critical quality parameter specific to visual tracking is *tracking precision.* If the precision is compromised, the objects lose track and the tracking process fails.

If we examine the internal behavior of the application, both CPU and network bandwidth resources affect tracking precision. First, since trackers are computationally intensive, the *tracking frequency* defined in Section III depends on CPU availability. Second, even if CPU is available to satisfy a required *tracking frequency*, *frame rate* in the video streaming implementation is still affected by network bandwidth availability. Divided by resource type, we identify available adaptation choices, including parameter-tuning and reconfiguring options, as the following.

• *Adaptation choices related to bandwidth.* First, there exist options for parameter-tuning adaptations (data adaptation) in the case of uncompressed image transfer, such as: (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 is, the higher the probability that the objects move out of the range. (2) 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. (3) The *frame rate* can also be altered, however in our application we attempt to minimize the usage of this option because the trackers are very sensitive to large variations of the frame rate. Second, if we consider reconfiguration choices (functional adaptation), *compression* and corresponding *decompression* can be activated, using available choices such as Motion-JPEG and streaming MPEG-2. Bandwidth requirements are reduced dramatically at the expense of increased CPU load.

• *Adaptation choices related to CPU.* In the current implementation, there are three frequently used tracking algorithms: *Line* tracking and *corner* tracking are edge based algorithms; and *SSD* tracking is a region based algorithm. Measurements show that these algorithms present diverse computational complexity. Adding to the flexibility, the application can run multiple trackers on multiple objects simultaneously. These observations motivate the following reconfiguration choices: (1) Addition of trackers to utilize idle CPU; (2) Removal of running trackers to decrease CPU demand; (3) Replacement 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 each tracker in the feature detection stage of computation.

B.2 Design of Adaptor

The Adaptors in the distributed visual tracking application uses the PID control algorithm presented in the Task Control Model. There are two types of Adaptors: CPU and bandwidth Adaptor. The CPU Adaptor is executed based on observed *tracking frequency* values from the application showing CPU load. The bandwidth Adaptor is executed based on observed *frame rate* values, showing transmission throughput. The output of the Adaptors are used by the Configurator to determine the adaptation choices.

B.3 Design of Configurator

In the Configurator, the adaptation choices are expressed in the form of a *rule base* for the visual tracking application, following the generic form given in Equation (19). The Configurator accepts control values $u(k)$ from Adaptation Tasks in both the CPU and bandwidth Adaptors. Two linguistic variables, `frequency` and `rate` are used, corresponding to the CPU and bandwidth resource, respectively. Conceptually, fuzzy sets with respect to `frequency` are derived by the fuzzifier from the control values $u(k)$ produced by the CPU Adaptor, and can be interpreted as the acceptable *tracking frequency.* Similarly, `rate`, interpreted as data units transmitted per second, is related to the output of bandwidth Adaptor. We assume the range of measuring linguistic variable `frequency` is $[0, 50]$ with an unit of iterations per second, and the range of measuring `rate` is $[0, 2000]$ with an unit of kilobytes per second.

Two linguistic variables, `cpu_choice` and
tt rate_choice are used in the rule base, corresponding to the bandwidth adaptation and CPU adaptation choices, respectively. Examples of linguistic values for these linguistic variables are `compress`, `chopped_image` and `add_tracker`. The linguistic values used for both `frequency` and `rate` are `very_low`, `below_average`, `moderate`, `above_average` and `very_high`. Examples of various rules are shown below.

```
/* linguistic rules corresponding to bandwidth */
if rate is very_high
   then rate_choice is chopped_image
if frequency is very_high and rate is very_low
   then rate_choice is compress
if frequency is below_average and rate is above_average
   then rate_choice is RGB24_color
if frequency is below_average and rate is moderate
   then rate_choice is RGB16_color
if frequency is below_average and rate is below_average
   then rate_choice is grayscale
```

```
if frequency is very_low and rate is very_low
   then rate_choice is back_and_white

/* linguistic rules corresponding to cpu */
if frequency is very_high
   then cpu_choice is add_tracker
if frequency is below_average
   then cpu_choice is drop_tracker
if frequency is moderate
   then cpu_choice is replace_tracker
if frequency is above_average
   then cpu_choice is adjust_region
```

B.4 The Design of Membership Functions

In design practices of fuzzy control systems, Gaussian, triangular or trapezoidal shaped membership functions are used to define the linguistic values of a linguistic 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 11, in four universal sets for variables `frequency`, `rate`, `cpu_choice` and `rate_choice`, respectively.

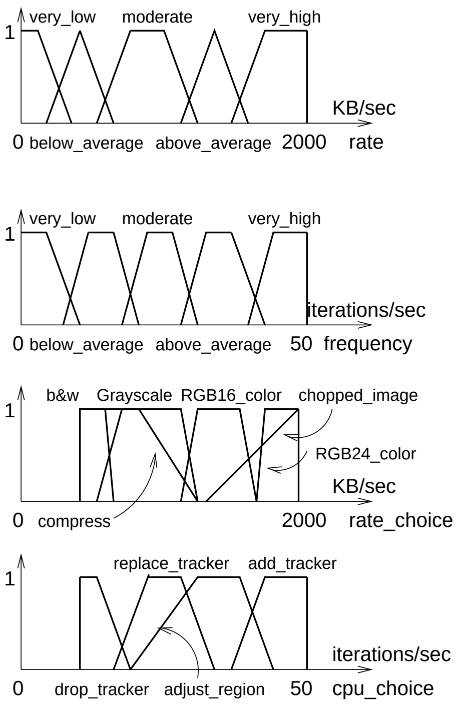


Fig. 11. Membership Functions in Visual Tracking

For example, if the input for `rate` is 50 KB/sec, and for `frequency` is 48 iterations/sec, the membership functions in Figure 11 show that the input linguistic value is `very_low` and `very_high` respectively. The output value from the Configurator will activate the adaptation choice `compress` determined by the corresponding membership function for `rate_choice`, according to the corresponding linguistic rule (`if frequency is very_high and rate is very_low then rate_choice is compress`).

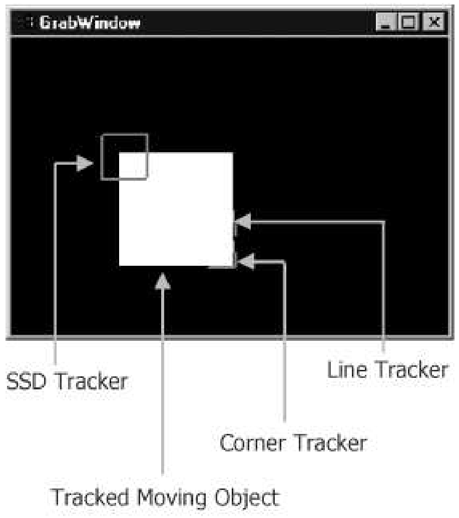


Fig. 12. A Running Client-Server Based Visual Tracking Application

B.5 The Flow of Adaptation Process

The distributed visual tracking application is controlled by the Middleware Control Framework based on current dynamics in CPU and bandwidth resources. Specifically, the adaptation process is carried out as follows.

- *CPU Adaptation*: The Observation Task in CPU Adaptor observes the current CPU resource usage in the application. Our observed parameter is *tracking frequency* (frequency of executing trackers). The Adaptation Task executes the PID control algorithm and produces new output values $u(k)$. If the tracking frequency ($u(k)$) drops below desired minimum, then the Configurator maps the results to actual adaptation choices, such as *drop tracker*. Finally, the adaptation choice is executed in the application, effectively changing CPU resource requirements.
- *Bandwidth Adaptation*: Since bandwidth is a partially controllable resource, the bandwidth Adaptor utilizes the extension to the Task Control Model in Section III-E. To observe the states in the Transmission Task, the Observation Task in bandwidth Adaptor at the server side monitors the current bandwidth usage in the application, by observing the throughput transmitted to the client. The client acknowledges all received data, so that the server-side Observation Task is able to calculate the amount of data in transit, treated as *outstanding resource requests*. The server-side Adaptation Task then completes the adaptation process, and the Configurator completes the mapping or nonlinear adaptation process to adaptation choices, such as *change image size* or *compress*. These choices are executed in the server application, effectively changing bandwidth resource requirements.

VI. Experimental Results

A. Deployment on Windows NT

For verification purposes, we have prototyped the client-server based distributed visual tracking application in Windows NT 4.0, porting and expanding the XVision program [12] on the Unix platform. To control the application, we have implemented the Adaptor and Configurator components within the Middleware Control Framework using C++. The Configurator is implemented with the assistance of the C-FLIE fuzzy inference engine [13]. The Observation Task is implemented in Java with visual feedback of current states to the user. Finally, we choose the CORBA 2.0 standard as the service enabling platform to facilitate interactions between the Middleware Control Framework and the visual tracking application. In our testbed we use ORBacus 3.1.1 for C++ and Java [14] as the CORBA 2.0 implementation. The application exports all possible adaptation choices to the middleware components as public IDL interfaces via CORBA, so that they can be activated when invoked by the middleware control components. This requirement demands that the application acts as a CORBA server to implement all public IDL interfaces and respond to incoming requests.

Our testbed of both distributed visual tracking application and middleware control framework is implemented on two dual Pentium Pro 200Mhz PCs running Windows NT 4.0. We use Visual C++ 6.0 as the primary development platform, and Sun Java 2 for Windows NT as the Java platform. We deployed both client and server on the same Ethernet segment.

B. *Experimental Results*

With respect to CPU load measurements, we perform both application-specific source-level instrumentation to obtain application-specific measurements such as *tracking frequency*, as well as system-level probes to measure the overall CPU load. The system-level CPU load data is measured by using the Performance Data Helper Library in the Platform SDK of Windows NT. In order to simulate the network bandwidth fluctuations, we plug in a throughput simulator to simulate a network through routers with FIFO packet scheduling and cross traffic. This setup simulates network fluctuations similar to what occur in the Internet. In addition, in order to measure the tracking precision and repeat experiments, we carry out all experiments based on animated moving objects served from the tracking server, instead of live video from the camera. Finally, we obtain the tracking precision by measuring the distance between the position of the tracker and the actual objects.

We experiment with three types of trackers that can run concurrently: the *SSD*, *corner* and *line* tracker. The *SSD* tracker is best used to track a moving outlined object, the *corner* tracker is best suited for tracking a corner of the object, and the *line* tracker is best suited for tracking lines and edges of objects. Figure 12 shows the GUI while the visual tracking application is being executed.

The first set of experiments focuses on parameter-tuning adaptation choices, which demonstrate the effectiveness of the Task Control Model. We show experimental results in Figures 13-20. These experiments use the bandwidth Adaptor, and focus on adaptation choices related to bandwidth requirements in the application. To focus on the purpose of the experiments, we use only one SSD tracker so that network throughput at the network device is the bottleneck affecting tracking precision. In this figure, the three graphs on the left show the case without activating the bandwidth Adaptor. The three graphs on the right show the case after adaptation. We can observe that by changing the image size of the visual tracking application, the tracking precision will be preserved without any tracking error during the connection lifetime. 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 verifying that the tracking algorithm loses the object. These experiments prove that the approach we have taken in the Task Control Model is effective in preserving tracking precision in a distributed environment with fluctuating bandwidth and significant end-to-end delay between the client and server.

The second set of experiments focuses on adaptation by reconfiguring choices, which emphasizes the effectiveness of the Fuzzy Control Model using both CPU and bandwidth Adaptors. We show results in Figures 21(a) and 21(b), Table II shows the output of the Configurator. In

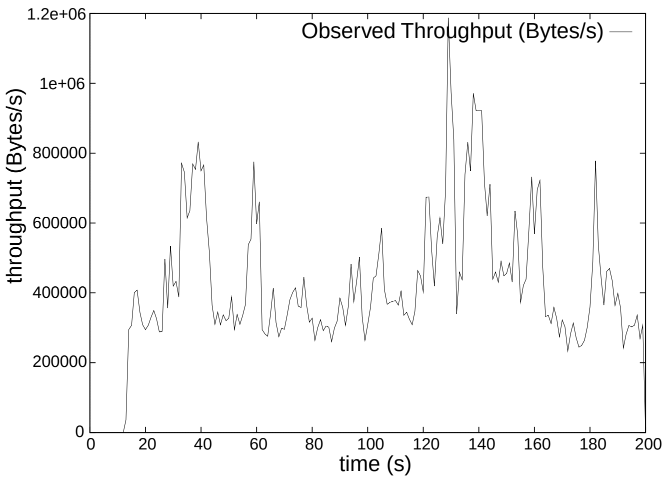


Fig. 13. Bandwidth Adaptor - Without Adaptation: Throughput

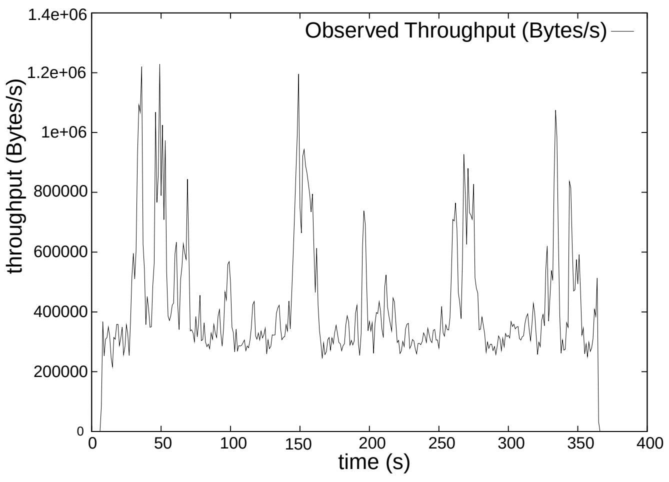


Fig. 14. Bandwidth Adaptor - With Adaptation: Throughput

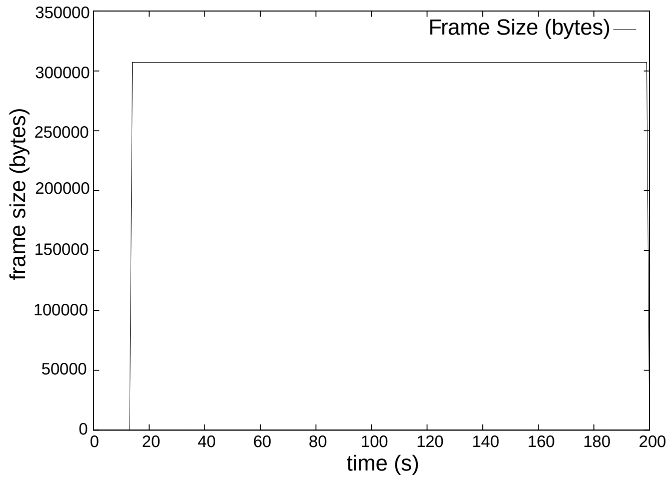


Fig. 15. Bandwidth Adaptor - Without Adaptation: Image Size

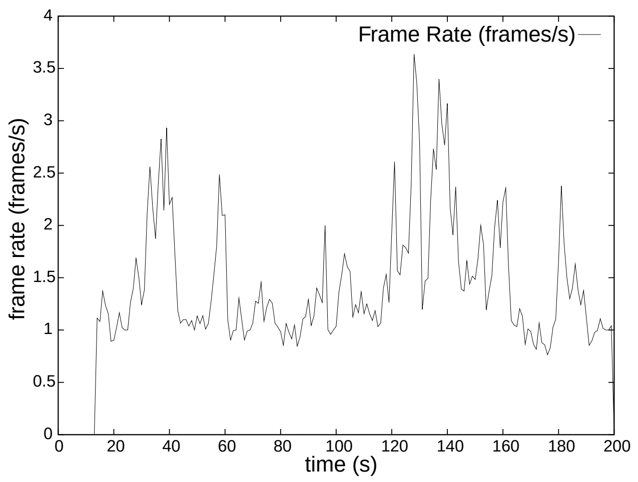


Fig. 17. Bandwidth Adaptor - Without Adaptation: Frame Rate

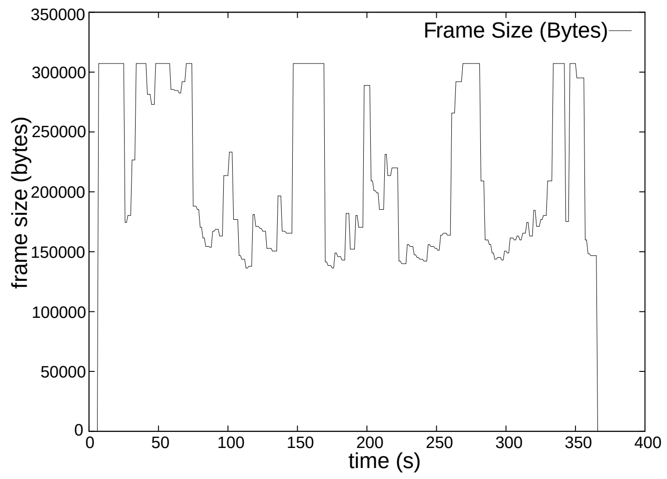


Fig. 16. Bandwidth Adaptor - With Adaptation: Image Size

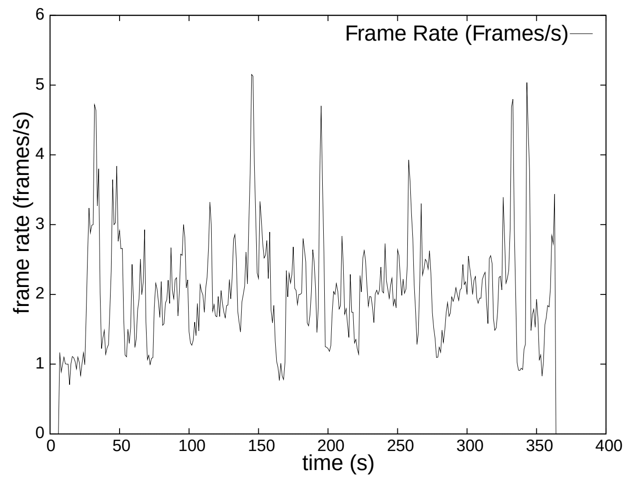


Fig. 18. Bandwidth Adaptor - With Adaptation: Frame Rate

this experiment, we focus on the CPU resource by using three concurrent trackers, SSD, line and corner tracker, concurrently. Figure 21(a) shows the CPU load measurements, while Table II shows the control actions generated by the Configurator at various starting times. Figure 21(b) shows the measured tracking precision. We assume that the first tracker tracks a more important object, so if a `drop_tracker` event is signaled, later trackers should be dropped. The results show that the tracking precision stays stable within a small range. This shows the effectiveness of CPU adaptations by a hybrid adaptation process in the CPU Adaptor and Configurator. In contrast, if reconfiguring actions are not activated, all trackers lose track when the CPU load increases in the end system.

C. Overall Evaluation

The results shown in the previous section experimentally prove the effectiveness of the Task Control and Fuzzy Control Model. On one hand, related to the network bandwidth, parameter-tuning adaptation choices are activated under the control of the bandwidth Adaptor, using theoretical results in the Task Control Model. On the other hand, reconfiguring choices such as `compress` are studied in a set of experiments studying the CPU Adaptor and the Configurator, using the Fuzzy Control Model. We conclude that in both cases, the proposed hybrid control approach is applied successfully to keep the critical quality parameter *tracking precision* stable, using a combination of linear and nonlinear adaptation process and an application-specific mapping process.

VII. Related Work

It has been widely recognized that many QoS-constrained distributed applications need to be adaptive in heterogeneous environments. Many research problems relevant to our work in the area of QoS adaptations have been studied. We briefly review each of them as follows.

A. Communication Protocols

System level adaptive mechanisms in communication and networking protocols have been extensively studied in the past decade. Particularly, the problem of flow control at the packet or cell level has been examined with great interests. *Flow control* refers to the set of techniques that

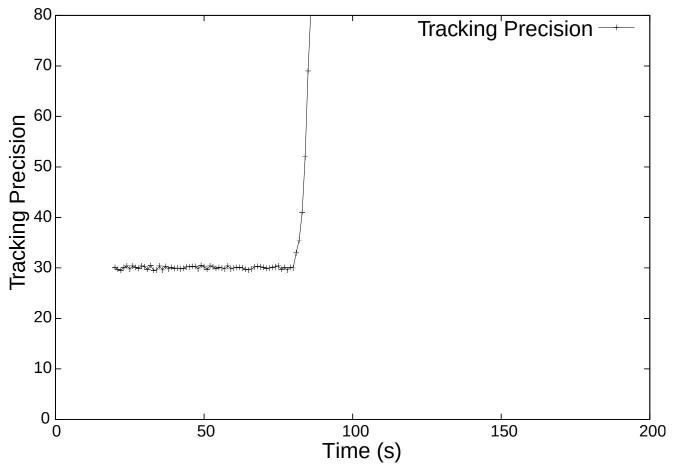


Fig. 19. Bandwidth Adaptor - Without Adaptation: Tracking Precision

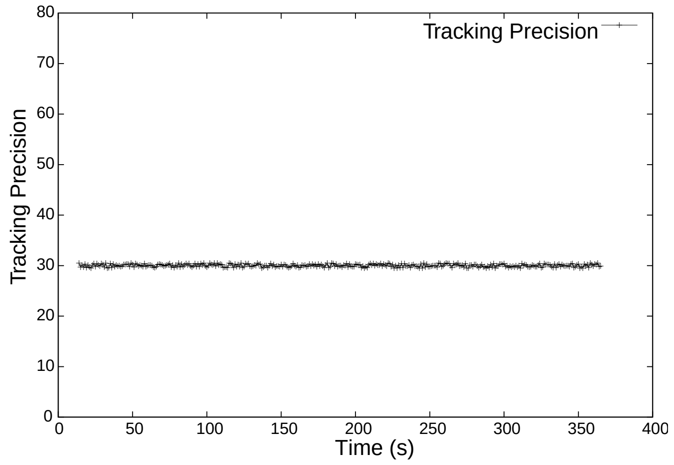
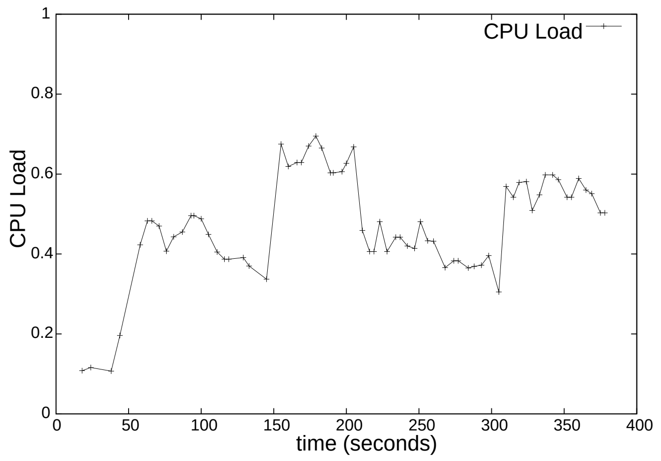
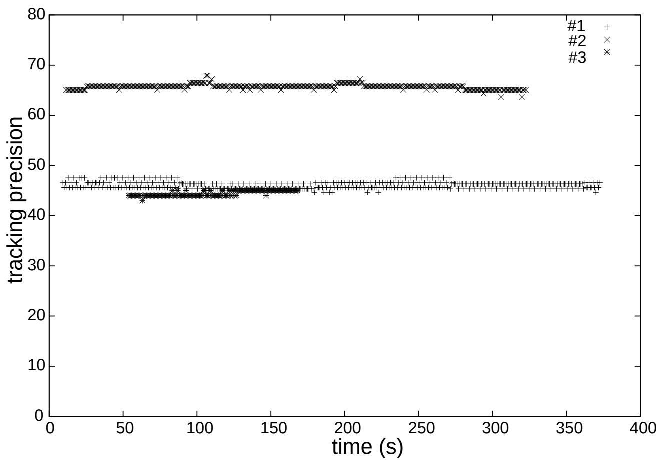


Fig. 20. Bandwidth Adaptor - With Adaptation: Tracking Precision



(a) CPU Load



(b) Tracking Precision

Fig. 21. Experimental Results with CPU-related Reconfigurations

enable a data source to match its transmission rate to the currently available service rate at a receiver and in the network. In this sense, the objective of the application-aware QoS adaptation with respect to network bandwidth resources is largely identical to the goal of the flow control, except that QoS adaptation is performed at the application level, and most mechanisms developed in the past decade related to the flow control are implemented at the datalink, network or transport layers of a protocol stack. We review some of the previous work related to our approach.

Many previous packet or cell level flow control approaches were proposed with the assistance of control theory. Earlier work [8] [15] [16] showed that a control-theoretic way of analyzing flow control problems is both valid and feasible. Notably, Keshav in [8] proposes a packet-pair flow control algorithm and uses control theory to analyze its stability and performance, under the assumption that a round-robin like scheduling discipline referred to as the *Rate Allocation Server* is deployed in the bottleneck node. In this work it is shown that the flow control algorithm is *stable*, which implies that if a new source becomes active, existing active sources adjust their transmission rates so that after a brief transient period, the system settles down to a new equilibrium. More recent work [9][17][18] [19] mainly focuses on the flow and congestion control issues for ATM switches, particularly under ABR (Available Bit Rate) service. Notably, in [17], the control laws for congestion control, in the case of a single congested node, were derived and stability properties proved. In [9], the stability and sensitivity properties are analyzed in the case of the rate-based flow control for ABR service.

Previous work has also studied the application of fuzzy logic and fuzzy control systems. In [8] fuzzy logic was applied to solve state estimation problems. Pitsillides et al. [13] present a fuzzy control approach 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 the active control of distributed multimedia applications, with efforts to adapt best to the environment. Furthermore, in the *AutoPilot* [20] project, a fuzzy logic approach

Start Time (sec)	*Control Action from Configurator*
28.22	`uncompress`
51.24	`add_tracker`
67.37	`compress`
167.7	`drop_tracker`
320.4	`drop_tracker`

TABLE II
Control Actions generated by the Configurator

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.

Our work focuses on the study of adaptations in the application domain, and adaptations with respect to more than one type of resource. While the mechanisms are certainly different in the application domain, the general approach of utilizing the control theory and the definitions for stability and other transient properties are similar to the previous work. Furthermore, we focus on global fairness properties of the adaptation behavior in an end system, which was not studied in most of the previous work on packet or cell-level flow control based on control theory.

B. Resource management

Recent research work on resource management mechanisms at the systems level also expressed much interests in studying various kinds of adaptive capabilities. Particularly, in wireless networking and mobile computing research, because of resource scarcity and bursty channel errors in wireless links, QoS adaptations are necessary in many occasions. For instance, in the work presented in [1], a set of adaptive resource management mechanisms was proposed that apply to the unique characteristics of a mobile environment. These adaptive mechanisms include the division of services into several service classes, predictive advanced resource reservation, and the notion of cost-effective adaptation by associating each adaptation action with a minimal lost of network revenue. Furthermore, Bianchi et al. [21] present a utility-fair allocation scheme implemented by a centralized adaptation controller, which is similar to our approach with respect to definition of fairness and the centralized approach to design adaptation strategies.

Another example is the work of Noble et al. in [5], who investigate an application-aware adaptation scheme, focusing on two characteristics: *fidelity* of data and *agility* of adaptation. Similar to our work, this work also builds on the separation principle between adaptation algorithms controlled by the system and application-specific mechanisms addressed by the application. This work is different from our approach since applications handle notifications and upcalls from the system, and adapt by themselves.

Another related category of work studies the problem of dynamic resource allocations, often at the operating systems level [22] [23][24]. The work in [23] focuses on maximizing the overall *system utility* functions, while keeping QoS received by each application within a feasible range (e.g., above a minimum bound). In [22], a global resource management system is proposed, which relies on middleware services as agents to assist resource management and negotiations. In [24], the work focuses on a multi-machine environment running a single complex application, and the objective is to promptly adjust resource allocation to adapt to changes in application's resource needs, whenever there is a risk of failing to satisfy the application's timing constraints.

In contrast to the above related work, our work distinguishes in its domain, focus and solutions. First, our work on the Task Control Model focuses on the analysis of the adaptation process, which is more natural for modeling with a control-theoretic approach, rather than overall system utility factors. Second, rather than focusing on a multi-machine environment running a single complex application, our work focuses on an environment with multiple applications competing for a limited amount of shared resources. Third, our work focuses on proposing various algorithms and models for the middleware components to actively control the application, rather than providing resource allocation and management services in the execution environment to meet the application's needs. In other words, we focus on assisting to adapt applications, rather than on resource allocations in the system.

C. Middleware Services

In addition to studies in the networking and resource management levels, many active research efforts are also dedicated to various adaptive functionalities provided by middleware services. For example, [11] proposes real-time extensions to CORBA which enables end-to-end QoS specification and enforcement. [25] proposes various extensions to standard CORBA components and services, in order to support adaptation, delegation and renegotiation services to shield QoS variations. The work applies particularly in the case of remote method invocations to objects over a wide-area network.

Our work is orthogonal and complementary to the above approaches, since the middleware control framework is based on underlying service enabling platforms, which is CORBA in our experimental testbed. In addition, we attempt to provide adaptation support to the applications proactively, rather than integrating adaptation mechanisms in CORBA services so that they are provided transparently to the applications. Furthermore, we develop mechanisms that are as generic as possible, applicable to applications with various demands and behavior. Finally, we provide support in the middleware control framework with respect to multiple resources, notably CPU and network bandwidth.

D. Application-specific Mechanisms

Recent research efforts are also particularly interested in adaptation problems in the application level. For example, the work presented in [26] and [3] uses software feedback mechanisms that enhance system adaptiveness by adjusting video sending rate according to on-the-fly network variations. In [27], Hafid et al. propose application 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. In [28], a software framework is proposed for network-aware applications to adequately adapt to network variations. [29] and [30] propose adaptive filtering mechanisms to reshape video streams, performed in either end systems or intermediate nodes in a multipeer distributed environment. In the work of Goktas et al. [31], 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. This is similar to our work that attempts to apply control theory to analyze the adaptation dynamics in a broader range of applications, and in a more rigorous fashion. In [32], a control model is proposed for adaptive QoS specification in an end-to-end scenario. In the work of Bolot et al. [33], rate and error control schemes are proposed at the application level for a video conferencing application in best-effort networks, also utilizing a scheme similar to rate-based feedback control.

Similar to the above, our approach models applications as a series of tasks, assisted by the feedback loop. However, we differ in our approach of using middleware components to control the adaptation behavior of applications, so that properties such as fairness can be derived in the end systems. Furthermore, the algorithms proposed to determine timing of adaptation in most previous work are heuristic in nature, and the analysis of various adaptive transient properties such as stability and agility are not addressed formally.

VIII. Conclusions

In this work, we presented a middleware control framework as a viable approach to reason about and model the dynamic control of QoS adaptations in flexible distributed applications. This framework includes two major components. The *Adaptor* follows control algorithms based on the Task Control Model. The *Configurator* takes output produced by the *Adaptor* and makes decisions on adaptation choices based on the Fuzzy Control Model. Both are implemented as middleware components and take advantage of standard service enabling platforms such as CORBA to control the applications. Analytical and experimental results show that this framework improves the adaptation awareness and effectiveness of flexible applications with respect to the preservation of critical quality parameters, while the adaptation choices are highly flexible and configurable according to the needs of individual applications. We are able to reason about and validate such properties as stability, agility, and equilibrium fairness in the adaptation process, which was not possible in previous work. Furthermore, our experimental results using the distributed visual tracking application convincingly validate our analytical results and show the feasibility and practicality of deploying flexible applications under the control of the middleware control framework.

Acknowledgments

We would like to thank Won J. Jeon, Jun-hyuk Seo, Kihun Kim and Mukul Chawla for their contributions to the work, as well as the reviewers for their most helpful and constructive comments to improve the presentation of this paper.

Appendix A: Proof of Equilibrium Fairness

Proof: Let x_s and $\bar{u}_s$ be the equilibrium values corresponding to the system established by Equation (9) and (12).

$$x_s = \Psi_{C_{max}}\{x_s + l_s\bar{u}_s + R_s - c\} \tag{20}$$

$$\gamma_i\, l_s\, \bar{u}_s = \Psi_{r_i}\, \{\gamma_i\, l_s\, \bar{u}_s + \alpha(x_s^c - x_s)\} \tag{21}$$

Ignoring the threshold cases, the solution to Equation (20) and (21) is

$$\bar{u}_s = \frac{c - R_s}{l_s} \tag{22}$$

$$x_s = x_s^c \tag{23}$$

Equation (23) directly proves the first part of the theorem. Assume the stable set of adapted tasks is A_s, Equation (22) can be rewritten for task T_i at equilibrium as follows:

$$\begin{aligned}(u_i)_s = \gamma_i\, l_s\, \bar{u}_s &= \frac{w_i\, l_s}{\sum_{T_j \in A_s} w_j} \frac{c - R_s}{l_s} \\ = \frac{w_i\, l_s}{\sum_{T_j \in A_s} w_j} &\left\{ \frac{c}{M_s} + \frac{(M_s - l_s)\frac{c}{M_s} - R_s}{l_s} \right\}\end{aligned} \tag{24}$$

where $(u_i)_s$ is the stable equilibrium that $u_i(k)$ converges to.

Equation (24) presents the following *weighted max-min fairness* property. Each task T_i can be granted at least a w_i share of the resources. In addition, if $M_s - l_s$ tasks request less than their fair share, namely, only l_s tasks are adapted, then the free portion $\frac{(M_s - l_s)c}{M_s} - R_s$ can be distributed among those Target Tasks which are degraded and thus need these resources. The distribution can be done according to their static weights w_i, which identify their relative priority and importance. This concludes the proof. ■

Appendix B: Proof of Asymptotic Stability

Proof:
We define

$$e(k) = x^c(k) - x(k) \tag{25}$$

$$\hat{u}_i(k) = \gamma_i \; l(k) \; [\bar{u}(k) - \frac{c - R(k)}{l(k)}] \tag{26}$$

In order to examine the asymptotic stability properties, we simplify the dynamic equations (9) and (12) in the neighborhood of equilibrium by: (1) removing the nonlinearities introduced by $\Psi_b(a)$ at both thresholds; (2) treating $l(k)$ and $R(k)$ as constants in the neighborhood of the equilibrium. Thus, Equations (9) and (12) become:

$$\hat{u}_i(k) = \hat{u}_i(k-1) + \alpha \; e(k) + \beta \; [e(k) - e(k-1)] \tag{27}$$

$$x(k) = x(k-1) + \frac{1}{\gamma_i}\hat{u}(k-1) \tag{28}$$

In order to derive the stability conditions, we perform z-transform on Difference Equations (27) and (28) to obtain $D_i(z)$ and $G_i(z)$, respectively. The transfer function $F_i(z)$ of the entire system is [6]:

$$F_i(z) = \frac{D_i(z)G_i(z)}{1 + D_i(z)G_i(z)} = \frac{\frac{1}{\gamma_i}[(\alpha+\beta)z - \beta]}{z^2 + (\frac{\alpha}{\gamma_i} + \frac{\beta}{\gamma_i} - 2)z - (\frac{\beta}{\gamma_i} - 1)} \tag{29}$$

We then consider the discrete characteristic equation of the above:

$$z^2 + (\frac{\alpha}{\gamma_i} + \frac{\beta}{\gamma_i} - 2)z - (\frac{\beta}{\gamma_i} - 1) = 0 \tag{30}$$

According to theorems in the digital control theory [6], in order for the system to be stable, all roots of Equation (30) need to be within the stability boundary, which is the unit circle. In other words, for any root z, we need $|z| < 1$. It can be proved that this property holds if the following condition is valid (the proof is omitted for space limitations):

$$\alpha > 0, \; \beta > 0, \;\; \text{and} \; \alpha + 2\beta < 4\gamma_i \tag{31}$$

Equation (31) concludes the proof. ■

Appendix C: The Fuzzy Inference Engine

The fuzzy inference engine operates by using the dual concepts of generalized modus ponens and compositional rule of inference [13] [10].

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:

$$\mu_{B^*}(y) = \sup_x[\mu_{A^*}(x) \star \mu_A(x)\mu_B(y)] \tag{32}$$

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:

$$\mu_{A_1^{(k)} \times \ldots \times A_n^{(k)}}(x_1, \cdots, x_n) = \mu_{A_1^{(k)}}(x_1) \star \cdots \star \mu_{A_n^{(k)}}(x_n) \tag{33}$$

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 (32) and (33). The calculation is as follows:

$$\mu_{B^{(1)*} \cup \ldots \cup B^{(m)*}}(y) = \mu_{B^{(1)*}}(y) \otimes \cdots \otimes \mu_{B^{(m)*}}(y) \tag{34}$$

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

Appendix D: The Fuzzification Process

The mapping process in the fuzzy inference engine calculates fuzzy sets as results, taking fuzzy sets as inputs. The calculated union of fuzzy sets $\tilde{B}^{(k)*}(k = 1, \ldots, m)$ is the output of the inference engine, while the fuzzy set $\tilde{A}^*$ is the input. 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 $\tilde{A}^*$.

With known input x_{in}, if there is no uncertainty in the numerical values, a simple fuzzification process can be:

$$\mu_{A^*}(x) = \begin{cases} 1, & \text{if } x = x_{in} \\ 0, & \text{if } x \neq x_{in} \end{cases} \tag{35}$$

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 (32) will become

$$\mu_{B^*}(y) = \mu_A(x_{in})\mu_B(y) \tag{36}$$

In the case of multiple input variables, we substitute Equation (33) in (36) and obtain

$$\mu_{B^*}(y) = \min[\mu_{A_1}(x_{in}), \cdots, \mu_{A_n}(x_{in})]\mu_B(y) \tag{37}$$

to compute the output of one inference rule. Finally, we compute an overall decision by applying Equation (34) to aggregate the calculated $\tilde{B}^{(k)*}, k = 1, \ldots, m$. This shows that the simple fuzzification process shown in Equation (35) simplifies the inference process in the inference engine.

Appendix E: 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. An example, the *Center of Gravity* method divides the integral of the area under the membership function of the output fuzzy set (Equation 37) into half, and the defuzzified value y_{out} marks the dividing point. Formally in the continuous case, this results in

$$y_{out} = \frac{\int y\mu_{B^*}(y)dy}{\int \mu_{B^*}(y)dy} \tag{38}$$

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. `chopped_image` associated with *image size*) and the tuning range $[p_{min}, p_{max}]$, then the modified value of p is set at

$$p = (p_{max} - p_{min}) * \mu_B(y_{out}) + p_{min} \tag{39}$$

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

References

[1] V. Bharghavan, K.-W. Lee, S. Lu, S. Ha, J. Li, and D. Dwyer, "The TIMELY Adaptive Resource Management Architecture," *IEEE Personal Communications Magazine*, vol. 5, no. 4, August 1998.

[2] H. Chu and K. Nahrstedt, "CPU Service Classes for Multimedia Applications," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems 1999*, June 1999, vol. 1, pp. 296–301.

[3] Z. Chen, S. M. Tan, R. H. Campbell, and Y. Li, "Real Time Video and Audio in the World Wide Web," *World Wide Web Journal*, vol. 1, January 1996, available from http://choices.cs.uiuc.edu/Papers/Vosaic/vosaic.pdf.

[4] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing," in *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing*, May 1996, available from http://www.cs.cmu.edu/afs/cs/project/coda/Web/docdir/podc95.pdf.

[5] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker, "Agile Application-Aware Adaptation for Mobility," in *Proceedings of the 16th ACM Symposium on Operating Systems and Principles*, Saint-Malo, France, October 1997, available from http://www.eecs.umich.edu/ bnoble/papers/s16.ps.

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

[7] D. Hull, A. Shankar, K. Nahrstedt, and J. Liu, "An End-to-End QoS Model and Management Architecture," in *Proceedings of IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, December 1997, pp. 82–89.

[8] S. Keshav, "A Control-Theoretic Approach to Flow Control," in *Proceedings of ACM SIGCOMM '91*, September 1991, pp. 3–15.

[9] W. Tsai, Y. Kim, and C-K Toh, "A Stability and Sensitivity Theory for Rate-based Max-Min ABR Flow Control," in *Proceedings of 6th IEEE Singapore International Conference on Networks*, June 1998, available from http://www.eng.uci.edu/ netrol/publication/SICON98-100.ps.

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

[11] D. Schmidt, D. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," *Computer Communications Journal*, vol. 21, no. 4, April 1998.

[12] G. Hager and K. Toyama, "The XVision System: A General-Purpose Substrate for Portable Real-Time Vision Applications," *Journal of Computer Vision and Image Understanding*, vol. 69, no. 1, pp. 23–37, 1997.

[13] A. Pitsillides, Y. A. Sekercioglu, and G. Ramamurthy, "Effective Control of Traffic Flow in ATM Networks Using Fuzzy Explicit Rate Marking (FERM)," *IEEE Journal of Selected Areas in Communications*, vol. 15, no. 2, pp. 209–225, February 1997.

[14] Object Oriented Concepts Inc., "ORBacus for C++ and Java," *ftp://ftp.ooc.com/pub/OB/3.1/OB-3.1.1.pdf*, January 1999.

[15] R. Jain, *Control-theoretic Formulation of Operating Systems Resource Management Policies*, Garland Publishing Company, 1979.

[16] D.-M. Chiu and R. Jain, "Analysis of Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 17, pp. 1–14, 1989.

[17] L. Benmohamed and S. Meerkov, "Feedback Control of Congestion in Packet Switching Networks: The Case of a Single Congested Node," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 693–708, December 1993.

[18] S. Mascolo, D. Cavendish, and M. Gerla, "ATM Rate Based Congestion Control Using a Smith Predictor: an EPRCA Implementation," in *Proceedings of IEEE INFOCOM '96*, San Francisco, 1996, available from http://www.cs.ucla.edu/ dirceu/infosmith.ps.

[19] L. Benmohamed and Y.T. Wang, "A Control-Theoretic ABR Explicit Rate Algorithm for ATM Switches with Per-VC Queueing," in *Proceedings of IEEE INFOCOM '98, Session 2B*, 1998.

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

[21] G. Bianchi, A. Campbell, and R. Liao, "On Utility-Fair Adaptive Services in Wireless Networks," in *Proceedings of Sixth International Workshop on Quality of Service*, May 1998, pp. 256–267.

[22] 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, Special Issue on Operating System and Services*, 1998, available from http://www.htc.honeywell.com/projects/arm/papers/QoS_Distributed.ps.

[23] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," in *Proceedings of 18th IEEE Real-Time Systems Symposium*, December 1997, available from http://www.cs.cmu.edu/afs/cs/project/rtmach/public/papers/qos.ps.

[24] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha, "On Adaptive Resource Allocation for Complex Real-Time Ap-

plications," in *Proceedings of 18th IEEE Real-Time Systems Symposium*, December 1997, available from http://www.cs.gatech.edu/systems/papers/daniela/rtss97.ps.

[25] R. Vanegas, J. Zinky, J. Loyall, D. Karr, R. Schantz, and D. Bakken, "QuO's Runtime Support for Quality of Service in Distributed Objects," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, The Lake District, England, September 1998, available from http://www.dist-systems.bbn.com/papers/1998/Middleware.

[26] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole, "A Distributed Real-Time MPEG Video Audio Player," *Lecture Notes in Computer Science*, vol. 1018, pp. 151–162, 1995.

[27] A. Hafid and G. Bochmann, "Quality of Service Adaptation in Distributed Multimedia Applications," *ACM Springer-Verlag Multimedia Systems Journal*, vol. 6, no. 5, September 1998, available from http://www.csd.uwo.ca/ faculty/hakim/papers/MMSystems.ps.

[28] J. Bolliger and T. Gross, "A Framework-Based Approach to the Development of Network-Aware Applications," *IEEE Transactions on Software Engineering, Special Issue on Mobility and Network-Aware Computing*, vol. 24, no. 5, pp. 376–390, May 1998.

[29] A. Campbell and G. Coulson, "QoS Adaptive Transports: Delivering Scalable Media to the Desk Top," *IEEE Network Magazine*, vol. 11, no. 2, pp. 18–27, March 1997.

[30] N. Yeadon, F. Garcia, D. Hutchison, and D. Shepherd, "Filters: QoS Support Mechanisms for Multipeer Communications," *IEEE Journal on Selected Areas in Communications, Special Issue on Distributed Multimedia Systems and Technology*, vol. 14, no. 7, pp. 1245–1262, September 1996.

[31] F. Goktas, J. Smith, and R. Bajcsy, "Telerobotics over Communication Networks," in *Proceedings of 36th IEEE Conference on Decision and Control*, San Diego, California, December 1997, pp. 2393–2399.

[32] J. DeMeer, "On the Specification of End-to-End QoS Control," in *Proceedings of the Fifth International Workshop on Quality of Service*, May 1997, pp. 195–198.

[33] J-C. Bolot and T. Turletti, "A Rate Control Scheme for Packet Video in the Internet," in *Proceedings of IEEE INFOCOM '94*, Toronto, Canada, June 1994, pp. 1216–1223.

Baochun Li received his B.Engr. and M.S. degrees in computer science from Tsinghua University, Beijing, P.R. China, and University of Illinois at Urbana-Champaign, respectively. Currently he is a PhD candidate in the department of computer science at University of Illinois at Urbana-Champaign. His research interests include quality of service, application-aware adaptation, multimedia networking, resource management, and distributed computing. His email address is `b-li@cs.uiuc.edu`.

Klara Nahrstedt (M' 94) received her A.B., M.Sc degrees in mathematics from Humboldt University, Berlin, Germany, and Ph.D. in computer science from the University of Pennsylvania. She is an assistant professor at the University of Illinois at Urbana-Champaign, Department of Computer Science, where she does research on Quality of Service (QoS)-aware systems with emphasis on end-to-end resource management and middleware issues for distributed multimedia systems. She is the coauthor of the widely used multimedia book *Multimedia: Computing, Communications and Applications* published by Prentice Hall, and the recipient of the Early NSF Career Award and the Junior Xerox Award for Research Achievements. Her email address is `klara@cs.uiuc.edu`.