System Support for Time-Critical Applications

J. Duane Northcutt and Eugene M. Kuerner
Sun Microsystems Laboratories, Inc.

A number of interesting application areas have time constraints associated with them — most notably in the area of multimedia. Today's workstations do not provide adequate support for applications whose definition of correctness is a function of time. This paper presents a set of system-level mechanisms that permit the incorporation of multiple streams of sustained, high-bandwidth, time-critical information as first-class data types within a distributed workstation computing environment. This work is being done as a part of a broader research effort being conducted at Sun Microsystems Laboratories to develop fundamental system technology in support of multimedia applications.

Introduction

The primary objective of this work is to develop system-level mechanisms which support the integration into the workstation of applications that manipulate time-critical information (e.g., audio and video [Hopper 90]). The definition of correctness for this class of applications is a function of time. It is not sufficient to simply perform a computation, it must be completed by a specific point in time for it to be of maximum value to the application [Northcutt 87].

The management of application timeliness requirements is one key concern. The ability to express the timeliness constraints of various activities and to temporally coordinate (i.e., synchronize) activities within, and among, application programs are the major issues that must be addressed. Furthermore, whenever a system is called upon to support activities with associated time-constraints, it is necessary to address the cases where the system lacks sufficient resources to meet all of the given time-constraints.

This paper describes a portion of a larger effort that is attempting to introduce the notion of time-criticality into the distributed workstation programming environment. The emphasis of the work described here is on the creation, and validation through use within a larger system context, of a programming interface that supports the needs of time-critical applications within the workstation. This interface will provide the means for expressing an application's timeliness requirements to the system (to permit the system to manage its resources so that application time constraints can be met) as well as the application's overload handling policies (so that the system can properly deal with the case where the application's time constraints cannot be met).

What is called for is the active management of resources by the system in order to meet the expressed time-constraints of its instantaneous application mix. In effect, the system must close the control loop of activities that have timeliness constraints associated with them.

This perspective implies that time-critical applications must interact with the system in a fundamentally different way than they do today. In particular, different information (in both kind and amount) must be exchanged between the system and its users, and the control relationship between applications and the underlying system must be significantly different (e.g., involving asynchronous notifications, up-calls, etc.).

Problem Statement

There is an entire class of problems to which today's workstations cannot be applied [HRV 90]. This application area is characterized by the inclusion of information or activities whose correctness (or value to the user) is a function of time. Multimedia audio and video, visualization, virtual reality, transaction processing, and data acquisition are examples of this class of applications. Current workstation system architectures fail to support time-critical applications for three main reasons. The system does not have the proper resources, the resources are not correctly organized, and the system resources are not managed in the proper fashion. This work is aimed at augmenting workstation functionality in order to allow applications to integrate time-critical information into the workstation.

Time Regulation of Application-Level Activities

In this paper, time-critical activities are defined to be operations that involve information that has time constraints associated with its manipulation. The manipulation of time-critical information is defined to include the following types of activities: *acquisition* (i.e., getting information into the system); *processing* (i.e., performing computations on the information); *transport* (i.e., moving the information between sets of sources and destinations); *storage* (i.e., buffering the information for later access); *coordination* (i.e., regulating the time when other forms of manipulation are performed on the information); and *presentation* (i.e., the delivery of the information to its end-user).

For the purposes of this work, individual units of time-critical information are referred to as *samples*, while sequences of such units are known here as *streams*. Each sample has a time constraint associated with it, and each manipulation of the data in the end-to-end sequence inherits its time constraints from the samples currently being dealt with. Thus, some activities are inherently time-critical (i.e., they are involved in the manipulation of time-critical information), while other activities acquire time-criticality through their association with other time-critical activities (i.e., if a function is performed on behalf of a time-critical activity, the function must inherit the client's time constraints).

In addition to meeting the timeliness needs of individual activities, it is necessary for the platform to support application-specified temporal alignments of separate time-critical activities. For example, the audio and video streams in a teleconferencing application require one type of time coordination, while the cueing of audio and video tracks in a compound multimedia document calls for another type of coordination. It is essential that the system support many different forms of temporal relationships [Steinmetz 90]. These range from strong forms of temporal alignment, where samples occur at specific times (e.g., generate an image when a word is clicked in a document), to weaker forms where samples of the constituent streams occur at given rates (e.g., generate audio, video, text according to a scripted presentation).

Key Technical Objectives

The most critical issues guiding the solution to the problem of time-critical application support are, high utilization of system resources, graceful degradation of functionality in the face of insufficient resources, and modularity in the expression of applications time constraints.

A high degree of system resource utilization is considered essential, because workstations traditionally provide best-effort services and applications areas such as multimedia are essentially open-ended in terms of resource demands. In effect, the system objectives should be to deliver the highest degree of value to the users that is possible, given an instantaneous mix of application demands and available system resources. This precludes the use of techniques for meeting timeliness needs that rely on excess assets [Liu 73].

For similar reasons, the notion of graceful degradation is considered quite important to solving timeliness problems. With respect to its time-critical behavior, the system will need to provide a range of different quality-of-service levels. In cases where there are insufficient resources to meet application time constraints, the system may shed some of its load to reclaim system resources or degrade the quality of service provided to an application. The specifics of such policy decisions should be left to the application to define, and the system should provide support for a wide range of such policies.

In addition, the notions of modularity and extensibility are of great value. Each application should be able to dynamically express its instantaneous time constraints to the system. In turn, the system should make every effort to deliver the greatest amount of value to the users as is possible. While global knowledge and cooperative programming will almost always result in greater overall effectiveness, the system should require that each application express only its own timeliness needs. Furthermore, timeliness mechanisms must also permit the dynamic interconnection of applications with arbitrary topologies and the transparent insertion of new functions into time-critical information manipulation paths.

Research Context

This paper describes the system interface component of a whole-system approach to bringing support for time-critical applications to the workstation. Given the strong degree of interaction frequently required between the various components of a multimedia system (and the relatively unknown nature of this application domain) it has been valuable to develop an overall system framework within which various components could be developed. The following is a brief description of this framework.

The *Time-Critical Applications Layer* exists at the same layer as ordinary workstation applications, where both time-critical and ordinary applications will coexist within a dynamic, multitasking, distributed environment. Time-critical applications will typically not generate explicit time constraints, nor usage policies, but will instead make use of programming packages that provide abstracted interfaces to the system's time-critical services.

The *Application-Specific Programming Packages* provide interfaces that are familiar to (or specialized for) the practitioners within a specific application area (e.g., audio and video editing, compound document authoring, etc.). The software packages at this level provide the top-level policies for the management of specific types of time-critical information (e.g., media), and do not expose the distinct entities that are involved in the management of timeliness, nor do these packages require the programmer to express its explicit time constraints to the system.

The *Device Virtualization Layer* is responsible for abstracting out many of the limitations of physical devices and providing the higher layers of software with the desired logical view of the underlying system. This layer manages mapping of virtual to physical devices — i.e., it handles (perhaps transparently) the multiplexing of logical devices onto physical ones. These new abstractions are not necessarily specific to a particular type of media but provide generic abstractions for dealing with the issues related to the management of time-critical information. Time issues are much more concrete at this layer, and this is where much of the policy for time-critical activities resides. Users of this layer express desires for more abstract behaviors of virtual resources (e.g., "multi-way interactive"

versus "stored unidirectional" channels of video), which are used in conjunction with this layer's policy decisions (e.g., permissible degree of jitter or delay, definition of exceptions, desired exception behavior, etc.) to generate the specific parameters required by the underlying software layers.

The *Operating System Layer* consists of two sub-layers: the kernel portion and the system portion. Both of these are logically part of the operating system. They form the basic functionality of the system as it is delivered and are separated from higher layers of software by some form of protection.

The *System Sub-Layer* is where the work described in this document is focussed. These are new system-level programming abstractions supporting time-critical applications. These abstractions are responsible for translating the more abstract requests for the manipulation of time-critical information into the specific parameters needed by the kernel-level mechanisms to ensure that system timeliness constraints are met. The programming abstractions also serve to direct manipulations of time-critical information into stylized forms that the operating system can manage. Furthermore, these programming abstractions concentrate timeliness-related concerns into a common mechanism, as opposed to requiring that concerns for timeliness be distributed throughout the entire operating system interface. This level implements the user-selected policies that provide the remaining information required by the kernel-level mechanisms to meet the more abstract timeliness needs expressed by this level's clients.

The *Kernel Sub-Layer* is where the management of the fundamental system resources (e.g., processor, memory, i/o, etc.) is performed. In order for the system to meet the demands of time-critical applications, it is critical for the resource management decisions to be performed in accordance with the time constraints of the computations making the individual resource requests. To accomplish this, a technique known as Time-Driven Resource Management (TDRM) is being used [Northcutt 88] [HRV 91a]. This approach requires that time constraints be associated with each time-critical activity, as well as directives as to what should be done in the event that the constraints cannot be met. This approach is quite different from traditional ones which attempt to encode the orthogonal attributes of importance and timeliness into a single value (typically, known as process priority) and involves quite different interactions between the operating system and its users.

Technical Approach

The following describes the major technical decisions that lie behind the definition of the system-level timeliness mechanisms and some of the major system-level implications of these decisions.

Key Design Decisions

First and foremost, the decision was made to develop a set of programming abstractions to encapsulate all notions of timeliness at the system level, and communicate specific timeliness requirements to the operating system kernel. In addition, the decision was made to view all issues related to the management of time-constraints as manipulations of time-critical information (i.e., a data-flow, as opposed to a control-flow perspective). This structures all manipulations of time-critical information into stylized activities that can be managed by the operating system. Finally, a model of time and synchronization was adopted to provide a framework for specifying the entire scope of temporal relationships that time-critical activities may assume (both individually, and with respect to one another) and the policies that might be followed in dealing with overload conditions.

Programming Abstractions for Timeliness

Two new programming abstractions are proposed here. First, a mechanism is provided to encapsulate the endpoints of time-critical information streams. This mechanism provides a uniform interface to a wide variety of physical, as well as logical, sources and sinks of time-critical information. These objects encapsulate the state associated with each endpoint entity includes such characteristics as: the maximum, minimum, and nominal sample rate (e.g., 44.1kHz, 30 Frames/second, etc.); the units of sampling granularity (e.g., video fields, video frames, blocks of audio samples, etc.); the amount of buffering provided; and the current state of the unit (e.g., running, stopped, buffer over-/under-flow, etc.). Secondly, active entities are provided to regulate the manipulation of the time-critical information streams flowing between the (passive) source and destination entities. This regulation involves the exercise of control over the time at which individual samples are acquired, stored, transported, processed, and presented.

The manipulation of time-critical information is regulated by a pair of logical activities that exist within the time-regulation entity. The first of these activities is involved with the acquisition of samples from the stream's source object and the buffering of these samples within the regulation object. The other logical activity involves the movement of samples from the regulation object's buffer area to the destination object. Providing that the internal buffer neither overflows nor underflows, the regulation object can control the time at which the samples are delivered in accordance with the timing constraints associated with the information.

The objects that encapsulate time-critical information stream sources and sinks are known as **Transducers**, and the objects that serve to connect sources to sinks and regulate the flow of time-critical information samples are known as **Conduits**. (The concepts embodied by these mechanisms are similar to those described in [Herrtwich 90] and [Escobar 91].) Both of these objects are described in the following chapter (and in greater detail in [HRV 91b]), while the remainder of this chapter provides descriptions of the major issues surrounding the definition of these entities.

Data-Flow-Based Approach

Integral to the foregoing definition of the timeliness programming abstractions is the point of view that all manipulation of time-critical information can be represented in terms of the timely transport of time-critical information samples.

This approach allows all user-level activities that have time constraints to be cast into a common form. This form is the timely transport of information from which all other manipulations (e.g., processing, storage, etc.) derive their time constraints. According to this model, all time-constraints stem from constraints on the time at which samples are delivered from the sources to the destinations.

The key technique in regulating time in this environment is the use of pre-fetching and buffering of samples in order to regulate their delivery times. Buffering can only smooth out transient timing disruptions (i.e., the phasing of sample delivery), so source and destination rates must be matched in the long-term. Therefore, higher level mechanisms are required to deal with long-term rate mismatches between connected sources and sinks of time-critical information.

The timely execution of processing activities is accomplished by the implicit propagation (or inheritance) of time constraints from the streams to the processing activities that operate on them. Computational elements are added to the delivery path of a time-critical information stream by encapsulating them within stream endpoint objects. The processing elements thereby acquire the necessary time constraint information.

Definition of Temporal Relationships

For the purposes of this work, there are two major forms of temporal relationships of interest. These are termed coordination and synchronization.

Event Coordination

Coordination is defined to be the general temporal alignment of events. This is a very general notion that involves only the act of ensuring that a set of events occur within a given interval of time with respect to each other. This alignment is not relative to any clock but is a function of the relative ordering of a set of events. In the case of time-critical information, coordination is viewed as part of the overall timeliness problem. Furthermore, all of the issues of time-critical information manipulation are combined into the single problem of ensuring that samples are delivered at the proper rates and phases with respect to each other.

Event Synchronization

Synchronization is defined here to be the temporal alignment of events relative to a physical clock. When time base is a real-time clock (or derived therefrom), then this conforms to the common notion of keeping an event stream flowing in real-time. In effect, synchronization is a special case of the more general notion of event coordination. As with coordination, there are both inter-stream and intra-stream synchronization issues. Aside from the addition of a real time base, all of the issues related to coordination apply to synchronization.

Generally speaking, a stream of time-critical samples can be considered synchronized if the samples are delivered to the destination at a specified rate. By defining synchronization criteria in terms of rates (as opposed to absolute time periods), it is possible to avoid the imposition of unnecessarily strict timing constraints on the underlying system. That is, rate-based synchronization does not require that each sample in a continuous stream be delivered at a specific time from the start of the stream. Rather, rate-based synchronization requires only that a given number of samples be delivered in some unit of time. This definition allows the users to express only that degree of synchronization required of their specific application. This provides the system with the least restrictive set of time constraints that are appropriate for the application mix.

Major Implications

In general, the support of time-critical applications requires new forms of interaction between applications and system software. These new interactions include the exchange between the application and the system of information concerning the timeliness constraints of individual computations (e.g, the amount of execution time required over the next real time interval), as well as policy direction as to what should be done in the event that the time constraints cannot be met.

Furthermore, overload (i.e., the case when resource requests exceed the system's current supply) will be a common case in typical multimedia systems (as opposed to an exceptional one). Therefore, the system must provide mechanisms for efficient interactions between the system and the applications. When resource requests cannot be satisfied, the requesting entities are to be notified. It is up to the client of these services to choose how and when notifications of such conditions are to be performed.

Time Regulation Mechanisms

This section describes the external view of the basic objects and the means by which they are combined into useful higher-level structures. In addition, a high level description is given of how time constraints are specified in practice and of the time-regulation mechanisms' internal structure and behavior.

External View

Basic Mechanisms

As mentioned earlier, the two new programming abstractions provide for the support of time-critical applications are known as *transducers* and *conduits*. These abstractions are combined in various ways to provide practical time regulation objects.

Transducers

Transducers are the endpoints in a time-regulated delivery chain. They are passive entities that provide uniform access to a diverse set of potential sources and sinks of time-critical information. Much like device drivers, transducers achieve their uniform appearance through the use of a common interface for supplying and consuming data. Transducers appear as system objects (i.e., abstract data types), have unique identifiers ("handles"), encapsulate state information, and have operations defined on them.

The operations defined on transducers include control-oriented (e.g., start, stop, pause, or resume), and data-oriented (e.g., get a sample from the source, or put a sample into the destination) ones.

In addition to encapsulating a logical source or sink of time-critical information, transducers include state information that reflect the timeliness characteristics of the enclosed entity. Conduits, the active time regulation objects, use transducer state information to implement time-regulated information transport.

Conduits

Conduits actively regulate sample delivery from source to destination. The conduit object encapsulates buffer space for samples, state information associated with the sample stream instance, and a regulation unit that oversees the desired time regulated information transport. As with transducers, conduits have a standard set of operations defined on them, which can be invoked by referencing the unique identifier (handle) associated with each conduit.

In addition to control oriented operations (e.g., start, stop, pause, resume) and state get/set operations, the conduit interface includes a number of connection points for asynchronous signals. Particularly, clients (users or other objects) may register interests or notifications for certain conditions that occur during the conduit's operation. These signals are used to notify clients of status, aberrations, and exceptions. Clients that temporally align (coordinate) multiple conduits make use of these signals. Additionally, conduits have asynchronous signal inputs that allow for dynamic adjustments to the conduit's (instantaneous) sample rate.

Compositions of Mechanisms

In order to create useful time regulation units from these basic abstractions, it is necessary that they be composed into a complete unit. Conduits and transducers may be combined both in simple and arbitrarily complex ways.

Basic Constructions

A simple composition is one in which a source transducer is connected to a destination transducer via a basic conduit. Such a construction permits a time-critical sample stream between the source and destination transducers to be regulated according to the intersection of user defined requirements, system limitations, and transducer requirements.

Complex Constructions

Objects which result from complex compositions of conduits and transducers project the same interface as basic compositions and are functionally indistinguishable to the programmer. However, the creation of complex compositions does involve the creation of additional regulation activities that implement independent stream coordination.

The system supports the two following composition types: "serial" compositions and "parallel" compositions. A serial composition is the connection of a set of complex or basic constructions together into a pipelined logical stream. A parallel composition is a hierarchical combination of concurrent time-regulated streams.

The ability to create pipeline-like compositions with time-critical information streams is useful for the same reasons that byte streams, pipes, and filters are so powerful in the UNIX operating system. Serial constructions allow for transformations on single data streams by embedding processing into the time-regulated information stream. Using serial constructions, embedded hardware or software processing elements inherit the stream's time-constraints, becoming time-critical processing elements that manipulate the information. Furthermore, as in the case of message passing in distributed systems, the serial composition of streams makes the physical node boundaries of machines transparent to clients.

Logically independent time-regulated streams that require coordination are realized with parallel constructions. A practical example of such a construction would involve the combining of a pair of (left and right) audio channels into a single logical (stereo) audio stream. This logical audio stream could then be combined with a video channel to create a movie entity. In this way, the logical streams encapsulate all of the necessary coordination information so that they can be used in the same way as primitive elements. Similarly two or more entities can be composed into a higher level one (e.g., four audio channels for quadraphonic audio). Finally, the compositions need not be at the same level of abstraction (e.g., stereo audio, a composition itself, is combined with a primitive video stream).

Time Management

Due to this work's data-flow orientation and emphasis on the multimedia application domain (specifically, continuous media [Anderson 91]), the time regulation mechanisms focus on managing the delivery times of samples within a stream of time-critical information. This model is used even when the activities that require time-regulation consist of aperiodic, asynchronous, or sporadic events.

Synchronization

Time-critical information is regulated according to the notion of minimally constraining (or "weak") synchronization. Weak synchronization recognizes that different applications have varying requirements with regard to the manipulation of time-critical information. For example, applications may be able to specify average rates over some interval, rather than demand hard time constraints. Therefore, the operating system has more flexibility in managing system resources and maximizing system utilization.

Basic constructions take advantage of weak synchronization, when possible, by defining a synchronization interval over which some number of samples are to be transported from the source to the destination transducers. If the number of samples is (N) and the synchronization interval is (T), then the regulation efforts of the basic construction focuses on transporting N samples every T units of time. The flexibility of this approach is derived from the fact that the system makes no guarantees regarding when during the interval T these N samples are delivered. The only presumed guarantee is that the average rate N/T samples/second is maintained across each time interval T.

It is worth noting that applications are not required to define time-regulated transport according to weak synchronization. The model permits application requests that result in strong (versus weak) synchronization. The transport of one sample during a given short interval of time represents such a hard limit.

Software Phase-Locked Loops

The time-regulation and subsequent transport adjustments basic objects make are done through a software phase-locked loop mechanism. The analogy to hardware phase-locked loops is very strong. The regulation activity controls the rate at which samples are emitted from the regulation mechanisms. This rate is adjusted based on information (an error signal) generated by comparing the actual output of the stream to the desired output (which can be varied by an external signal) over a given time interval. The integration period defined by the evaluation time interval acts as a low pass filter in the mechanism's feedback path. This helps ensure stability of the control mechanism and imposes more manageable demands on the underlying kernel mechanisms. The regulation mechanisms generally emit samples at a nominal rate and periodically adjust their phases in an attempt to follow a reference event stream.

Much as with hardware phase-locked loops, there is a continuous clock that is used as the free running reference (i.e. the time-base input), and there is another signal used to speed up or slow down the stream. This provides rate adjustments derived as some time delta from the base frequency. The regulated streams attempt to track changes in the reference event stream by speeding up or slowing down sample delivery. In the event that the reference rate departs from the defined "capture range" of a time regulated stream, the regulation mechanism enters a free-running state. The samples are transported at the stream's nominal rate (or "center frequency"). In this way, lost or delayed reference events do not disrupt the flow of samples from time-regulated streams.

Mechanism Internals

Basic and complex constructions encapsulate some amount of state and a set of logical management activities. Additionally, their active nature requires that they generate and accept asynchronous events and carry out specific policies when certain predefined conditions occur.

Local State Information

A major portion of the state information encapsulated by a time regulation object is comprised of the sample buffer (when one exists). The size of this buffer area is defined in terms of the maximum number of samples that can be stored at a given time. This value is specified at instantiation time and can be dynamically modified (within bounds) at run-time. The buffer area is managed as a FIFO queue, and the instantaneous number of samples in the queue is included as part of the object's state information. The occupancy of the buffer is critical to the maximum effectiveness of the time regulation mechanism.

Therefore, state information is kept which defines the minimum, maximum, and expected occupancy values for the buffer (i.e., the high-water, low-water, and nominal queue depths). Note that the latency imposed by the mechanism is proportional to the depth of the sample queue. In order to support the limit case of minimal latency (at a cost in variations in inter-arrival time or sample jitter), it is possible to specify no buffering for a basic construction.

In addition to the buffer area and its parameters, a number of other parameters are encapsulated by the regulation objects. Some of these are parameters that define the minimum, maximum, and expected rates of the sample producers and consumers, some are statistics accumulated by the system to reflect the current state of the regulated stream (e.g., number of samples to be transferred in this time interval, number of sample actually transferred, etc.), and others are the external adjustments to the rate control process.

Activities

Three logical activities manage time-regulated sample transport between source and destination transducers in basic constructions. Within complex constructions, additional activities are introduced that manage the coordination between the underlying constructs.

Source Activity

The source activity is responsible for managing the source transducer and maintaining the input portion of the buffer (and its associated state). The source activity attempts to keep the nominal number of samples in the buffer. If the queue depth goes above the high-water mark, or overflows, the source activity signals the entities that have registered an interest in such events. Alternatively, if the buffer state indicates that samples are being transported faster than the source is currently producing them, the source activity may, if permitted, try to speed up source sample production to keep the input portion of the buffer at its nominal level.

Destination Activity

The destination activity manages the destination transducer and the movement of samples from the input portion of the buffer to the destination transducer. The destination activity's behavior is essentially identical to that of the source activity. It attempts to deliver samples to the destination at a rate specified by the regulation activity and signals queue under-run events.

Regulation Activity

The heart of time-regulated sample transport is the actual regulation activity. It uses the notion of a software phase-locked loop, described earlier, to evaluate sample transport progress and make adjustments as necessary.

Regulation is activated following each specified integration interval and the current state of the sample stream (i.e., how many samples have actually been delivered), is evaluated against the current desired number (i.e., how many samples should have been delivered — the number given by the user, plus or minus the instantaneous adjustment amount given by other things that the regulator is coordinating with). Based on the evaluation of the current time status of the stream (i.e., ahead, behind, or on-time), the regulation activity can issue modifications to the execution rate of the source and destination activities.

Composite Stream Coordination Activities

The serial and parallel coordination activities extend the regulation concept to a slightly higher level of abstraction. However, the same basic regulatory and adjustment technique is used. The complex construction regulation activities use the expected transport requirements of their components and

compares them to actual component sample transport rates during some defined interval. Depending on the comparison of the expected versus actual samples transported over a coordination interval (and the coordination policy), adjustments to speed up or slow down component transport rates are made.

Control Signals

During the course of normal execution, basic and complex regulation objects generate events to indicate interesting state changes to higher-level software and other regulated object in order to effect coordination among time-coordinated sample streams. Likewise, all objects are allowed to receive such signals and use them to adjust their sample transport timing. Current control signal categories include status conditions, aberration conditions, and exception conditions.

In addition, regulation objects receive an external reference signal that is used as a time-base. In the case of a single processing node, the time-base can be the same for all time regulation objects and may be derived from the system clock. In the physically distributed case, a logical common time-base can be provided (e.g., [Mills 90]). It should be noted here that the degree of time control that these mechanisms can exert on a stream can only be as accurate as the time-base that is provided.

It is possible to effectively "gate the clock" (i.e., selectively enable the delivery of time-base signals) in order to perform cueing operations or on demand sample delivery. This supports time-critical activities that depend on asynchronous events. For example, a button push event could be used as a time-base input to cause the regulation activity to request immediate resources for the short-term delivery of a sample. Likewise, these mechanisms can be used to deliver samples at specified queuing times.

Parameterized Policy

Run-time behavior is largely directed by user or client defined policy. The response of objects to status, aberration, and exception conditions is defined by default policies, or policies specified by the user.

Conclusion

The work presented here is a component of a larger system architecture. This layer is intended to provide an internal interface, that mediates interactions between applications and the operating system kernel. We know of no current alternative means of achieving the functionality provided by this facility. As such, it will be difficult to evaluate its effectiveness, but should be easy to find willing and eager test subjects for its use.

This paper describes work in progress, with an initial implementation currently underway. There will undoubtedly be many changes called for as these mechanisms are used in ongoing research and advanced development projects (both within and outside of Sun Microsystems). An effort is underway to identify and exploit opportunities for collaboration in this area in order to speed the development of an effective and broadly applicable facility.

References

[Anderson 91] Anderson, D. P., Govindan, R., and Homsy, G.
Abstractions for Continuous Media in a Network Window System.
International Conference on Multimedia Information Systems, Singapore, January 1991.

[Escobar 91] J. Escobar, D. Deutsch, and C. Partridge
A Multi-Service Flow Synchronization Protocol.
Technical Report, Bolt Beranek and Newman, March 1991.

[Herrtwich 90] R. G. Herrtwich
Time Capsules: An Abstraction for Access to Continuous-Media Data.
IEEE Real-Time Systems Symposium, December 1990.

[Hopper 90] A. Hopper
Pandora — An Experimental System for Multimedia Applications.
ACM Operating Systems Review, 1990.

[HRV 90] High Resolution Video (HRV) Workstation Project
System Support for Time-Critical Media Applications: Functional Requirements
HRV Project Technical Report #90101, November 1990.

[HRV 91a] High Resolution Video (HRV) Workstation Project
High Resolution Video Workstation: Executive Software Specification.
HRV Project Technical Report — in preparation, October 1991.

[HRV 91b] High Resolution Video (HRV) Workstation Project
High Resolution Video Workstation: Video Programming Abstractions.
HRV Project Technical Report — in preparation, October 1991.

[Liu 73] Liu, C. L. and Leyland, J. W.
Scheduling Algorithms for Multiprocessing in a Hard Real-Time Environment.
Journal of the ACM 20(1):46-61, 1973.

[Mills 90] Mills, D.
Network Time Protocol (Version 2) Specification and Implementation.
Network Working Group Technical Report, Delaware, July 1990.

[Northcutt 87] J. D. Northcutt
Mechanisms for Reliable Distributed Real-Time Operating Systems: The Alpha Kernel.
Academic Press, Boston, 1987.

[Northcutt 88] J. D. Northcutt
The Alpha Operating System: Requirements and Rationale.
Archons Project Technical Report #88011, Department of Computer Science, Carnegie-Mellon University, January 1988.

[Steinmetz 90] R. Steinmetz
Synchronization Properties in Multimedia Systems.
IEEE Journal on Selected Areas in Communications, 8(3), April 1990.

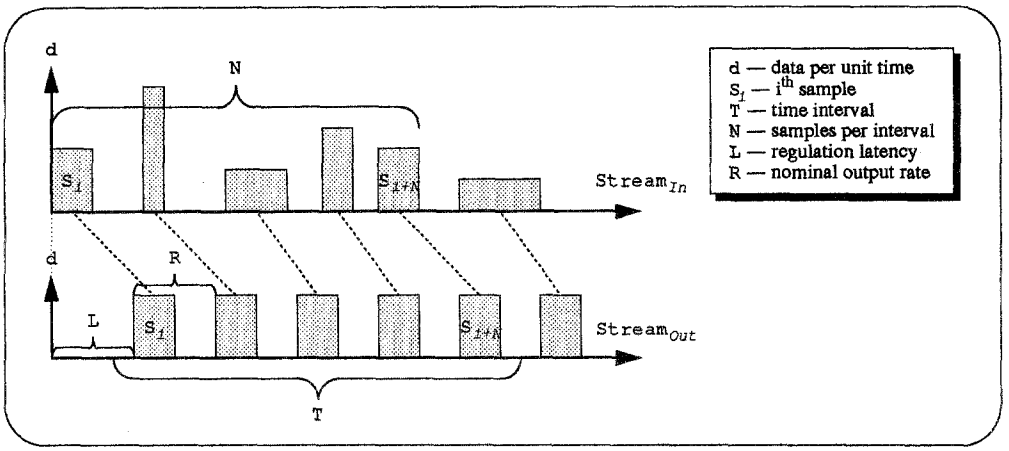

Figure 1 Regulation of Time-Critical Information

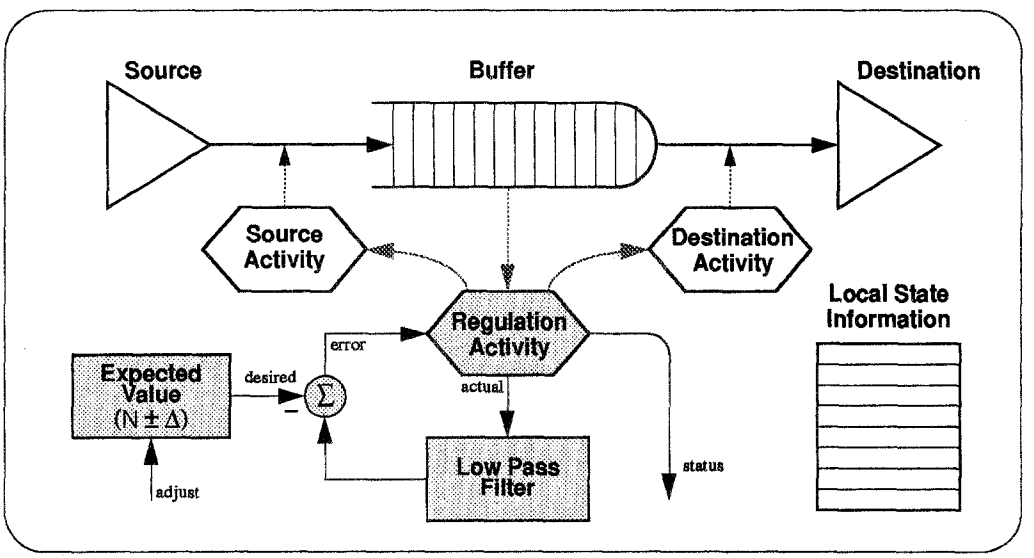

Figure 2 Software Phase-Locked Loop Regulation Mechanism