INTEGRATED DIGITAL CONTINUOUS MEDIA: A FRAMEWORK BASED ON MACH, X11, AND TCP/IP

David P. Anderson
Ramesh Govindan
George Homsy
Robert Wahbe

March 22, 1990

ABSTRACT

High-quality sound and motion video (''continuous media'') are potentially important elements of human/computer interfaces. Workstations will soon be commonly available with bus-based hardware for audio/video digital conversion and video compression/decompression, and will be connected by networks capable of handling continuous media traffic. This report describes an approach, called *integrated digital continuous media* (IDCM), to using continuous media in distributed computer systems. In the IDCM approach, continuous media data is handled like other data. It passes through system hardware (main memory, I/O bus, and CPU). User programs can input, output, process, communicate, store and retrieve continuous-media data in the same software framework (operating system, network protocols, window system) as other data types. Furthermore, such programs can run concurrently, sharing the resources of workstations, servers, and networks.

IDCM has many advantages over approaches that use separate facilities for storage and communication of continuous-media data. However, it raises many difficult system software design issues, ranging from real-time device scheduling to the design of user interfaces and programming toolkits. We enumerate and discuss these issues, and sketch the design of an IDCM software system that addresses many of them. Our design is based on industry-standard software components such as Mach, X11, and TCP/IP. It includes the Session Reservation Protocol (SRP) for distributed resource allocation and scheduling, and Continuous Media Extensions to X (CMEX), an extension of the X11 window system supporting IDCM.

1. INTRODUCTION

We use the term *continuous media*[1] to refer to the combination of sound and motion video[2] used as part of a human/computer interface. The use of continuous media in the computer/human interface represents a major step in the development of interface technology (see Table 1.1).

year	technology	paradigms
1950	punched cards, line printer	job
1960	hard-copy terminal	interactive login session
1970	cursor-addressable CRT	''forms'' applications, WYSIWYG text editors
1980	graphics, mouse	direct manipulation, WYSIWYG documents, virtual desktop
1990	sound, video	remote presence, virtual reality

Table 1.1
The evolution of user interface media.

From the point of view of hardware technology, the transition from CRT-systems to workstations with bitmap and mouse was fairly straightforward. It required combining a bit-mapped display (already used in CRTs) and a microcomputer (already used in personal computers). However, significant additions were needed in terms of software. Network-transparent window servers [11,20] and application-side user-interface toolkits [14] evolved gradually and with considerable discussion. However, this software placed few new requirements on operating systems or networks; it could be layered on existing systems such as MS-DOS and UNIX.

The transition from bitmap/mouse systems to continuous media is not as simple. This is because of the nature of these media: information must be produced or consumed at fixed (and high) rates. The raw data rate of digital video is too high for most subsystems (CPU, memory, network, disk). For both media, there are real-time requirements that cannot be met by existing operating systems, network communication systems, and programming paradigms.

Several approaches have been taken to deal with these problems. Systems can be placed in one of the three following categories according to how the CM data is represented and manipulated:

(1) CM data is stored and communicated in analog form, on machines under computer control, so that the continuous media can be integrated with discrete media. Examples include VOX [4] and IMAL [12].

(2) CM data is in digital form, but does not pass through the main memory of the computer. Examples include the Olivetti Pandora system [9] and the Xerox Etherphone [23].

[1] The term ''continuous media'' is used, rather than the overloaded term ''multimedia'', to emphasize the distinguishing characteristic of sound and video: that they change continuously over time. In contrast, text and graphics are *discrete* output media, and keyboards and mice are discrete input media.

[2] We group sound and motion video together because 1) they present the same general problems from the software point of view; 2) video is not generally useful without audio.

(3) CM data is in digital form and is handled in hardware by the primary memory and I/O systems, and in software by the OS and user programs. Examples include the Digidesign system for the Apple Macintosh [25], which handles only audio, and Intel's MS-DOS based DVI system [13], which handles video as well.

Of these approaches, the third has the advantage of *integration*. CM data can be manipulated algorithmically in the same way as other data, so the system is more flexible and general. Hardware may be simpler, since no separate disks or networks are required for CM data. However, existing systems in this category are severely restricted. They support only local applications, allow only one application to be run at once, and provide a limited and non-standard programming environment. We feel that an ideal software framework for continuous media should have the following properties:

(1) Users can run multiple applications concurrently, with no adverse affects from contention for hardware resources.

(2) Applications can be distributed, allowing continuous media data to be shipped in real time through digital communication networks.

(3) The software framework should include the essential elements of existing distributed environment: network-transparent window systems, network file access, naming and authentication, and so on. One way to achieve this goal is to build on an existing software environment such as that provided by UNIX (and its derivatives such as Mach), TCP/IP, X11 and its toolkits, and so on.

We say that a system with the above properties provides *integrated digital continuous media* (IDCM). This report is concerned with the software structure of IDCM systems. Our goals are to 1) argue for the feasibility and desirability of IDCM; 2) analyze the design problems at different levels of IDCM system software, and 3) propose some solutions to these problems.

The report is structured as follows. Section 2 gives examples of possible applications of an IDCM system, discusses the advantages of the approach, and describes the general requirements of IDCM applications. The subsequent sections of this report correspond to the levels of design of an IDCM system:

Section 3 discusses the problem of allocation and scheduling for IDCM. We describe *DASH resource model*, a unified approach to scheduling that allows users to create end-to-end "sessions" with the performance guarantees (delay and throughput) needed for continuous media.

Section 4 discusses operating system kernel design, and in particular how an operating system can be made to conform to the DASH resource model.

Section 5 discusses network communication and protocols. We describe the *Session Reservation Protocol* (SRP), which allows the DASH resource model to be incorporated in the DoD Internet protocol hierarchy.

Section 6 gives the design of Continuous Media Extensions to X (CMEX), an extension to the X11 protocol that supports digital continuous media as both input and output.

Finally, Section 7 gives conclusions and directions for further exploration.

2. ADVANTAGES AND REQUIREMENTS OF THE IDCM APPROACH

Compared with other approaches for handling continuous media in computer systems, IDCM has the following advantages:

- Digital CM data representation avoids the signal-quality loss inherent to analog systems when data is copied or communicated.

- All data in the IDCM system, including CM data, is handled (stored, retrieved, and transmitted) using the same hardware. There is no need for separate analog devices and networks, dedicated disks, and so on.
- The generality (*i.e.*, the Turing-completeness) of the computer applies to CM data. The ability to acquire, transform, and present data is restricted only by hardware performance, not by logical limitations.
- Because handling of CM data is integrated at the software level, the addition of CM functionality in existing applications (mail, documents, training) is simply a matter of extending the software. Furthermore, standard generic OS capabilities (such as ''teeing'' the output of a program into a file) can be augmented to apply to CM data as well.

IDCM imposes many requirements on the underlying system (OS, network, and protocols). Consider an application that provides a voice conversation between two people. Given our assumptions about the default hardware configuration in an IDCM system (see Section 6), this application requires only that we write software to manage the audio connection; no new hardware is needed.

This ''computer telephone'' application has several performance requirements. Each direction of the conversation may require throughput up to 1.4 Mb/s depending on quality. Furthermore, the allowable end-to-end delay must be less than a few hundred milliseconds (in some applications, such as distributed music rehearsal, the requirements are much more stringent). The IDCM system must be able to guarantee these requirements to the application. Glitches during the conversation are not acceptable. If a user runs a CPU-intensive application during the conversation, the sound quality should not suffer as a result. If such a guarantee is not possible, the system must inform the application so that appropriate action, such as asking for a lower quality connection, may be taken.

Assume now that both parties agree to create a recording of their conversation. How could such an application be provided? In an IDCM system it is done simply by duplicating the voice data in software, shipping one copy to the workstation speaker and the other to a file server to be saved in a file. A non-IDCM system would require special hardware such as a digital or analog recorder to be added to our workstation in addition to the software necessary to control the new device.

Extending our computer-telephone application to include live video increases the performance requirements. Uncompressed video ranges from .1 Gb/s to 5 GBits/sec. Currently available video compression systems produce data rates in the range from 64 Kb/s for ''conference quality'' to 3-4 Mb/s for NTSC quality. However, the software mechanisms used to guarantee performance for audio will also work for video. In particular, such a conversation can be recorded on a disk file in the same way.

In addition to these delay and throughput requirements, The audio/video conference requires synchronization between the audio and video ''tracks''. To provide ''lip synch'', the sound and video must be synchronized to within a few tens of milliseconds, or less. Other applications may have more complex synchronization requirements, including interactions of discrete and continuous media.

IDCM requires a new model for application program structure. An application might consist of a number of processes, one for each track of the audio and video and another for executing supervisory commands. The processes handling audio and video will each be reading or writing data to/from the network.

As a last example, consider a multimedia mail facility. Because CM data is stored in the same manner as text and graphics, we may easily compose mail messages which consist of different media. This would be quite difficult in a system in which video is stored, for example, on a analog video disk. How would one send a message composed of text and video (stored on this special purpose disk) across a wide area network? What if the receiver has a different hardware configuration?

The IDCM approach simplifies such applications for two reasons. First, it provides communication and location-independent storage (*i.e.*, network file servers) for CM data. Second, software layers can conceal hardware dependencies, possibly converting to and from standard CM data formats.

Incorporating CM into applications poses many new problems for software components such as the operating system and window system. As outlined in this section, these IDCM components must provide delay and throughput guarantees, precise timing control of processes, synchronization of multiple CM data streams, and mechanisms to accommodate hardware dependencies. However, once in place, such a system provides a versatile and powerful platform for CM applications.

3. THE DASH RESOURCE MODEL

As described in the previous section, IDCM applications have stringent end-to-end performance requirements. In typical applications, a stream of CM data may pass through many hardware devices (disk, memory, I/O bus, CPU, network links and switches, DSP chips, etc.). Each of these devices contributes to the end-to-end delay and can limit the end-to-end throughput. The devices will often be *shared* with concurrent applications or users, in which case they must be *scheduled* according to some policy.

The scheduling policies used in current general-purpose operating systems and networks are typically aimed at providing *fairness* rather than performance guarantees. Examples include the UNIX CPU and disk-head scheduling algorithms, the Ethernet media access protocol, and so on. Such scheduling policies cannot provide the guarantees needed for IDCM. The scheduling policies commonly used in real-time systems, on the other hand, are generally aimed at bounded response time for periodic arrivals, which is too restrictive for a general-purpose system.

We therefore have developed an approach to resource allocation and scheduling that we call the *DASH resource model* (described in more detail elsewhere [1]). In this model, the set of system components that handle continuous-media data is decomposed into a set of *resources*. A resource may correspond to a schedulable hardware device and its accompanying software driver. For example, a CPU and its scheduler can comprise a resource. Resources may also have a more complex structure: a wide area network (which includes multiple interface devices and data switches, concurrent operation, and multiple scheduling mechanisms) might be treated as a single resource.

Work is given to resources in units called *messages* (typically a segment of continuous-media data). Messages flow in unidirectional streams that pass through one or more resources. Data is generated by a *source* resource (a disk, digitizer, or compression unit), is then processed by a sequence of *handler resources* (networks, CPUs, etc.) and finally is consumed by a *sink resource* (disk, decompression unit, etc.).

The DASH resource model has the following components:

- Workloads (*i.e.*, streams of CM data) are described by a simple parameterization called a *linear bounded arrival process* (LBAP).
- Resources provide a standard interface allowing clients to create *sessions* with the resource. A session represents a reservation of part of the capacity of the resource. The client must specify its workload; in return, the resource provides a bound on the delay it will impose. These parameters determine the amount of buffer space needed to ensure freedom from buffer overrun.

- Sessions can be combined into *end-to-end sessions* spanning many resources. The division of end-to-end delay among resources is accomplished using an economic approach.

3.1. Linear Bounded Arrival Processes

The DASH resource model uses *linear bounded arrival processes (LBAPs)*, an abstraction introduced by Cruz [8]. An LBAP has the following parameters:

maximum message size	S_{max}	(bytes)
maximum message rate	R_{max}	(messages/second)
maximum burst size	B_{max}	(messages)

The long-term data rate of the LBAP is $S_{max} R_{max}$ bytes per second. In any time interval of length t, the number of messages arriving at the interface may not exceed $B_{max} + t\, R_{max}$. The burst parameter allows short-term violations of this rate constraint, modeling programs and devices that generate "bursts" of messages that would otherwise exceed the rate constraint. The *logical arrival time* of a message m is the time m would have arrived if the LBAP strictly obeyed its maximum message rate.

3.2. Sessions

A session has LBAP parameters describing its input and output arrival processes. In addition, a session has the following delay parameters:

maximum logical delay	L_{max}
minimum actual delay	A_{min}
maximum buffered delay	M_{max}

The *logical delay* of m is the interval between the m's logical arrival time and its logical arrival time at the output interface. The *buffered delay* is the portion of actual delay during which the message is stored in host memory.

When data traverses a sequence of resources, the *basic sessions* within the resources are said to form an *end-to-end session.* The output interface of each resource in an end-to-end session is the input interface of the next resource. Each resource must be prepared to handle the burst size generated by the previous resource. The *end-to-end logical delay* of a message is the interval between its logical arrival time at source output and its logical arrival time at the sink input. The maximum logical delay of an end-to-end session is the sum of the maximum logical delays of its handler resources.

3.3. Implementing the Resource Interface

The DASH resource model defines a uniform abstract interface to resources: clients request sessions, and the resource manager can grant or deny requests. The manager is responsible for honoring the delay and burst size limits for the requests it grants. If we restrict our attention to resources that consist of a single hardware device (*e.g.*, a CPU), we see that the resource interface can be successfully implemented on top of a range of scheduling policies. Any policy for which an upper bound on delay can be derived from given input LBAP parameters can be used. For example, *round-robin, FIFO, rate-monotonic* and *earliest-deadline-first* scheduling all have this bounded-delay property. Existing results in scheduling theory [7] can also be applied. Worst-case simulation provides a more general decision procedure [3].

For resources that encapsulate more than a single device, the management procedures are more complicated. For example, the sources of delay in an FDDI network (viewed as a single resource) include queueing, media access, and propagation, and many scheduling policies are possible. Establishing a session in such a resource may involve using network management protocols to reserve network bandwidth in "synchronous" or "isochronous" channels. The question of how to support sessions in such a resource is a subject of ongoing investigation.

3.4. An Economic Approach to Delay Allocation

To divide the delay between resources in an end-to-end session the DASH resource model takes an approach based on economics. When a client reserves a session with a resource, the resource makes reservations for the smallest possible maximum delay. In addition, the resource provides a *cost function* indicating, for each larger maximum delay, the associated cost to the client. For tractability, the DASH resource model requires that every cost function be 1) piecewise linear; 2) strictly monotonic decreasing, and 3) convex.

The cost of an end-to-end-session is the sum of the costs of its component sessions. cost functions to be piecewise linear convex functions, they can be combined by the following procedure: The segments of the functions are sorted in order of decreasing (more negative) slope. They are concatenated in this order, starting at the point which is the sum of the initial endpoints of the functions. The resulting function reflects the policy of returning excess delay in a way which minimizes total cost.

3.5. End-to-End Session Establishment

The DASH resource model defines an establishment protocol for end-to-end sessions. Using this protocol, the application's allowable end-to-end delay is divided between the resources, and burst sizes are established. The establishment protocol is carried out by *host resource managers* (HRMs). Initially, the HRM at the source host is given a client request that specifies the resources involved, the message size and rate, and the end-to-end logical delay requirements (a *target* and *maximum* value, denoted E_{target} and $E_{\max}$). The protocol has two phases:

(1) The first phase traverses the hosts from the source to the sink. A *request* message is relayed between HRMs. The request message contains the data message size and rate, the client delays E_{target} and $E_{\max}$, the burst size from the previous host, and the cumulative sums of $L_{\max}$ and $A_{\min}$, and the cumulative cost functions. Maximum reservations are made for each resource, and corresponding buffer space is reserved.

(2) The second phase proceeds in the reverse direction. The receiving client evaluates the end-to-end session parameters and decides on a delay for the session. A *reply* message containing the remaining excess delay (see below) and the burst size into the next host is passed back towards the source. For each resource, the session parameters are relaxed appropriately. The delay may be increased and additional buffers may be reserved both for this purpose and to accommodate larger input bursts. Delays are relaxed only up to the amount of buffer space available.

Let E_{actual} be the actual end-to-end logical delay obtained in the first phase of the session establishment. If this delay is less than E_{target}, some *excess logical delay* E_{excess} defined as $E_{\text{excess}} = E_{\text{target}} - E_{\text{actual}}$ can be distributed among the resources. This is typically done in the way that minimizes total cost.

4. OPERATING SYSTEM REQUIREMENTS

The basic issues of scheduling and reservation have been addressed in the previous sections. We will now explore the design of specific operating system facilities (process structure, timing, and virtual memory) as they relate to IDCM. At the center of our discussion is the question of how an operating system can conform to the DASH resource model in its scheduling of resources. For example, how can CPU execution be associated with sessions, and assigned the priority (or deadline) associated with that session? For a variety of reasons, this is sometimes impossible;

these situations are called *priority inversions.*

4.1. Internal OS Structure

Most versions of UNIX are nonpreemptible in the kernel. This means that if a higher-priority process becomes runnable during the execution of a system call, priority inversion occurs until the system call sleeps or returns. To avoid this situation it is desirable that the kernel be preemptible (this implies locking of data structures). Failing this, it is desirable that there be an upper bound on the duration of a system call, which can be factored into the delay guarantees for the CPU resource.

Internal synchronization (software-based locking of data structures) can lead to priority inversion. The duration of these inversions will be less than that caused by nonpreemptibility, and may be tolerable in some cases. Techniques such as the *priority ceiling* policy [21] for semaphore acquisition have been developed to deal with this problem. Priority inversion can also occur while interrupts are masked, since this can delay the handling of an interrupt that would awaken a higher-priority process. It is desirable that a bound on interrupts-masked periods be enforced.

4.2. User Process Structure

An IDCM program (client or server) will generally carry on multiple concurrent activities. For example, a client program might do the following concurrently: 1) transfer a digital sound stream from a file server to the media server; 2) transfer a digital video stream from a second file server to the media server; 3) wait for mouse or keyboard events from the media server. How should the programming of these activities be structured? There are several possible approaches (see Figure 4.1).

The simplest approach is for the program to consist of a single process that uses the UNIX `select()` system call (or its equivalent) to multiplex input sources (Figure 4.1a). This approach has the following drawbacks:

- There is no preemption of activities. If the processing of a mouse event uses a large burst of CPU time, the handling of the video stream would be delayed, producing an unacceptable glitch in the output.
- The approach leads to poorly-structured software. The main program has the form of an "event loop" that must be modified to add a new activity. The state of each activity must be explicitly encoded in a "state block" rather than in a process state (*i.e.*, call frame stack).
- On a multiprocessor architecture, there is no opportunity for true parallelism.

Of these problems, the first one (non-preemption) is the most serious; it effectively precludes the use of the single-process model. However, the single-process approach can be extended by allowing the process to use signals (user-level interrupts) to handle input (Figure 4.1b). This partly addresses the problem of non-preemption because, for example, the signal handler for video input could preempt (interrupt) the handler for mouse input. However, it is not an optimal solution in terms of scheduling. How should signal handlers be prioritized? The UNIX policy (in which all handlers have the same priority) cannot be used because it allows deadline inversion. Any fixed prioritization, in fact, allows deadline inversion. A policy that reflects deadlines would require each signal handler be given maximum priority, to evaluate its own deadline, to compare it with the deadlines of currently running processes, and possibly yield (using a software interrupt mechanism). This is not a clean solution.

A better solution is for the program to include several threads[3] sharing a single address

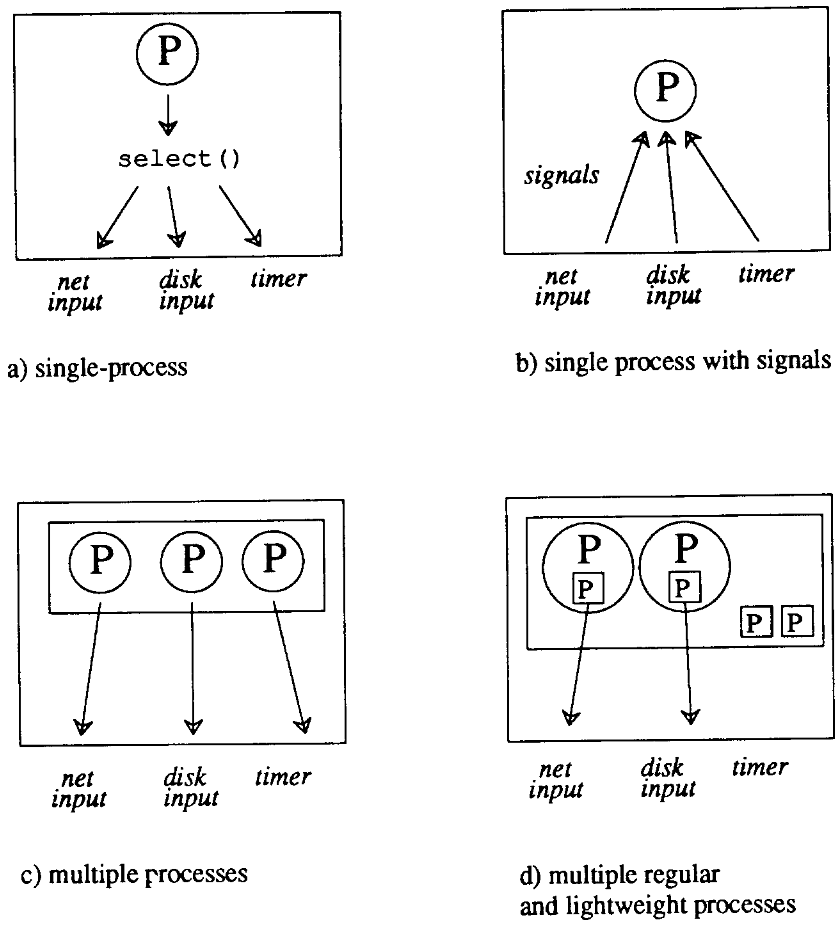


Figure 4.1
Alternative process structures for continuous media.

space (see Figure 4.1c). Typically one process is assigned to each concurrent activity. When an input event occurs, the corresponding process can be preemptively executed, if appropriate, with no unnecessary domain switches.

If this "multithreaded" approach is used, a new source of priority inversion arises: user-level synchronization. Since thread scheduling is preemptive, user-level data structures must be protected by locks (perhaps sleep locks accessed via system calls). If a low-priority thread acquires a lock and is then preempted by a high-priority thread that requests the same lock, a priority inversion of considerable duration is possible. This problem remains to be solved.

[3] A *thread* is a process scheduled by the operating system.

A final structuring possibility is to use lightweight processes (LWPs) per activity, with one "heavyweight" process (HWP) per CPU (Figure 4.1d). Each HWP process executes one LWP at any given point. Input events are presented as signals, and preemptive scheduling of LWPs can be done at the user level. This approach has been used, for example, in the Presto system [5]. Its advantages (which are not particularly relevant to IDCM) are that process creation and rescheduling are efficient, and that scheduling decisions can be made at the user level. This is a disadvantage for IDCM; for optimal scheduling among the possibly several programs running on a host, an LWP must inform the OS of the new priority of each LWP that it executes, requiring a system call for this purpose.

4.3. Time Control

IDCM programs need precise control over real time. Specifically, they need to obey LBAP parameters (obey burst size limit), maintain positive backlog, and generate timed output events[4]. Hence the programs need the following OS services:

(1) To read the current time (in microseconds) without a system call. For this purpose a readable microsecond clock can be mapped into the user address space.

(2) To sleep until a given real time (with a given scheduling deadline on wakeup), perhaps with the option of being awakened early if a message arrives.

The UNIX system calls `gettimeofday()`, `setitimer()`, and `select()` are not adequate for these purposes. They are oriented towards the "single-process with signals" process structure, and have insufficient precision (usually 0.01 seconds).

4.4. Virtual Memory

Paged virtual memory is an indispensable feature of modern operating systems. However, page faults at unpredictable times can preclude real-time guarantees. Hence IDCM applications may need to lock pages in physical memory. In the simplest approach, an entire address space would be declared as memory-resident. There may also be very large programs of which only a small part (those parts involved with transfer of CM data) need to be memory-resident. The identification of these parts to the OS may involved language-level support.

In our IDCM model, user-level programs handle CM data; they at least transfer it between I/O devices, and in some cases they may manipulate it directly. The efficiency of data movement between separate user VASs, and between the kernel and a user VAS, is a major consideration. The following general approaches are possible:

- CM data is copied by software, as in UNIX. This consumes CPU cycles and bus bandwidth, and may impose a bottleneck.
- CM data is stored in memory that is shared, read-write, among all VASs. This is maximally efficient, but may not provide adequate protection.
- CM data is moved by VM remapping between VASs. Examples include Mach [17], in which copy-on-write is used to provide duplication semantics, and data may change virtual address. The DASH kernel [24] uses a more efficient scheme in which transfer semantics are used, and data does not change address.

[4] Depending on the output hardware device, the timing of output may be determined by the device itself (*i.e.*, double-buffered audio that generates an interrupt on buffer empty, or video driven by retrace interrupts) or by the application program itself (unsynchronized video output, or animation).

5. NETWORK COMMUNICATION OF CONTINUOUS MEDIA DATA

This section discusses the problem of communicating IDCM data over digital communication networks. The fundamental problem is performance: the network must provide throughput and delay guarantees. Next-generation networks such as FDDI [19] and BISDN [6] are capable of such guarantees, and offer the raw performance needed for IDCM. The DASH resource model can provide the "glue" between these networks and the host operating systems. As part of the DASH project, we have developed the *Session Reservation Protocol* (SRP), a protocol for TCP/IP networks which can be used to implement the end-to-end session establishment algorithm.

A second network-related problem is the role of transport protocols. Standard transport protocols provide mechanisms (flow control and error recovery) that are not always needed by IDCM, and that may interfere with performance guarantees. We argue that, under certain conditions, these mechanisms are not exercised, and therefore these protocols can successfully be used for IDCM data.

5.1. Networks as Resources

The DASH resource model views an entire network, together with its host interfaces, as a single resource. A session in this resource provides a simplex communication path that begins with the call to the network output routine in the sending kernel and ends at the start of process-level handling in the receiving host. Even in simple cases (such as a single LAN) there are multiple sources of delay within such a resource:

(1) Local contention for the interface device (*i.e.*, output queueing);

(2) Media access (Ethernet collisions, waiting for token arrival, etc.);

(3) Transmission and propagation times.

In more complex networks there may be other sources of delay, such as switch contention. To cast a network in the DRM framework (*i.e.*, to implement the `reserve()` and `relax()` operations), all these sources of delay must be considered. The details of implementation depend on the network. For examples, in an Ethernet the local contention can be precisely controlled, but media access is unpredictable because of the random algorithm; it can be estimated or ignored, depending on the typical network loads. In an ISDN or BISDN network, the host is supplied with a contention-free connection (*i.e.*, contention is handled in the switching nodes, and is factored into the performance guarantees offered to customers). In an FDDI network, a host may reserve synchronous data or isochronous channels to limit its media access time. The assignment of outgoing messages to synchronous mode or to an isochronous channel is a policy of the network module.

5.2. The Session Reservation Protocol (SRP)

We have defined the Session Reservation Protocol (SRP) [2] so that performance guarantees can be made for communication based on IP [16]. SRP can be viewed as a "network management protocol" operating at the internetwork (IP) layer. SRP cooperates with the sending and receiving clients to implement the end-to-end session establishment algorithm of the DASH resource model. This end-to-end session is associated with a connection of a particular IP-based protocol (for example, a TCP connection). The performance guarantees of the end-to-end session apply to the data traffic from the sending to the receiving client on the associated connection.

SRP is responsible for reserving (and relaxing) the following resources :

- On the sending host, SRP reserves the network resource through which messages leave the host.

- On a gateway, SRP reserves the CPU resource needed for protocol operation and the network resource through which messages leave the gateway.
- On the receiving host, SRP makes no reservations, and simply conveys the session request to the receiving client. (The incoming network resource was reserved by the previous host in the path.)

The clients of SRP are responsible for reserving all other resources that handle the stream of continuous-media messages (for example, the reservation of an input or output device). Thus, on the sending and receiving hosts the role of the "host resource manager" is divided between the sending client and SRP.

SRP has the following properties:

- Independence from transport protocols: SRP can be used with standard protocols such as TCP or with new real-time protocols.
- Compatibility with IP: header fields of IP packets are not added or modified. On hosts that implement SRP, however, the IP module must be modified so that the relative priority of incoming packets can be established.
- A host implementing SRP can benefit from its use even when communicating with hosts (or through gateways) not supporting SRP.

The DARPA Internet framework provides both advantages and disadvantages for real-time communication. The datagram abstraction of the IP layer is compatible with continuous-media applications, which often do not require reliability and could not, in fact, tolerate the delays caused by retransmissions in reliable link-layer protocols. IP also has the advantage of widespread use. However, the dynamic routing generally used by IP implementations poses problems for real-time communication.

5.3. Transport Protocols for IDCM

In the presence of the DASH resource model, the functions of transport-layer protocols are simplified or eliminated. First, transport protocols are assigned the task of recovering from packet loss. This loss is usually due to network interface buffer overflow rather than bit errors (especially with optical technology). The DASH resource model eliminates buffer overflow for guaranteed sessions.

Second, transport protocols provide flow control (usually emphasizing end-to-end flow control, intended to avoid overflowing buffers in the receiving host. This type of flow control is not needed with the DASH resource model as long as sender obeys its contract. Clients may need fine rate adjustments (to compensate for differing clock speeds) but this is a different problem.

In short, the DASH resource model allows communication networks to be viewed like local hardware; *i.e.*, errors are very rare, and can be ignored. For some CM applications, errors (packet loss or corruption) may be tolerable. For audio, a lost or damaged packet produces a glitch in the sound, and likewise for non-differentially compressed video. For differentially-compressed video, the receiver may request a new complete frame or an update that includes the lost frame. If it making archive recording at same time, the application may request out-of-band retransmission.

TCP [15], the dominant stream transport protocol in IP networks, is not especially well-suited to continuous media applications. For continuous-media data, regular delivery is at least as important as reliable delivery, and sometimes more important. It is possible to use TCP for continuous-media data; however, its main features (flow control and error recovery) are not useful in this context. They may actually interfere with timing, and increase workload, in ways that are detrimental to real-time performance.

If clients obey the DASH resource model, they can use TCP as a "null" transport protocol. The TCP functions relating to addressing, multiplexing, and fragmentation are used, but flow control and retransmission normally are not.

6. CMEX: CONTINUOUS MEDIA EXTENSIONS TO X

In this section we describe a set of extensions to the X11 window system that allow an X11 server to handle continuous-media data. We call these extensions CMEX (Continuous-Media Extensions to X).

Our design assumes that the host on which the CMEX server runs (*i.e.*, the workstation) has hardware support for digital audio and/or video I/O in addition to the usual bitmap display, mouse and keyboard. For example, the workstation may have a video camera attached to the workstation and aimed at the user, and perhaps an additional camera for scanning nearby objects. These cameras are interfaced by video digitizers, and the workstation has a video DSP coprocessor for compressing this digitized video in real time. The hardware can also decompress digital video in real time, displaying the result within the bitmap display[5].

The workstation may also have hardware for audio input and/or output. The output might be via digital-to-analog converters to external speakers, a headset, or both. Input might be from a microphone or stereo microphone pair. The workstation might also have digital audio DSP hardware capable of mixing, sample rate conversion, filtering, and so on.

In this section we introduce CMEX via a set of graduated examples. The examples use simple scenarios to illustrate the concepts and mechanisms used in CMEX. The examples all involve output only; the corresponding scenarios for input are very similar.

6.1. Scenario 1 - Simple audio output

In the simplest scenario (see Figure 6.1), a client plays a monoaural audio stream through a speaker on the CMEX server. To do this, it first establishes a **continuous-media connection** over which the audio data is to be sent. It creates a **logical device** or LDev (a sort of "virtual speaker"), informs the server of the LDev's data format, and attaches the LDev to the CM connection. The client then **maps** the logical device to a physical speaker and begins sending data over the CM connection. The CMEX server converts this data to sound and plays it through the speaker.

We have introduced two concepts: the notion of a continuous media connection and the abstraction of a logical device (LDev). We discuss each of these in turn.

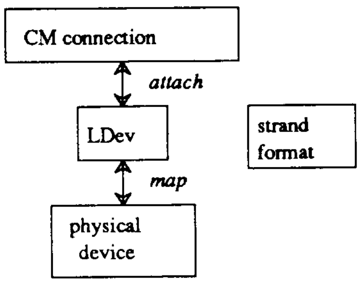
6.1.1. Logical Devices (LDevs)

A *logical continuous media device* (or LDev) represents a virtual CM hardware device. Multiple LDevs may be multiplexed onto a single physical device. LDevs may be of different *classes*: *players* and *listeners* for digital audio, *VWins* and *VCams* for digital video.

An LDev is a resource in CMEX. Some generic operations can be performed on LDevs. They are *create*, *destroy*, *map* and *unmap*. *Map* associates the LDev with a physical device. *Unmap* dissociates the LDev from the physical device. CMEX defines queries to find out the types of devices supported and their characteristics.

[5] Other hardware configurations are possible, though less convenient. For example, the workstation might have separate monitors for still graphics and video, or might have an overlay board allowing video to be mixed with still graphics.

server abstractions



messages

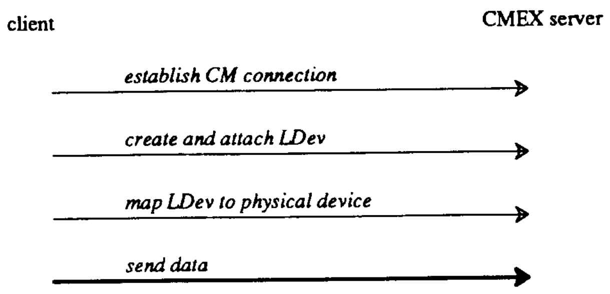


Figure 6.1
The abstractions and messages involvea in simple playback.

Every LDev has some associated attributes. The CMEX protocol defines requests for getting and setting the attributes of an LDev. Attributes of LDevs are of two types: *generic* attributes are associated with every LDev, while *specific* attributes vary with the class of the LDev. There are three important generic attributes of LDevs:

- *strand*: The strand attribute of an LDev indicates the representation of the data to be input from or output to the LDev. (see 6.2.1)
- *connection*: Every LDev has and associated CM connection over which data is sent or received (see Section 6.1.2).
- *LTS*: Each LDev has an associated LTS (see Section 6.4.1). CM data to or from the LDev is synchronized with respect to the logical time system of the device.
- *PDev*: This is the device number of the physical device to which the LDev is to be mapped. The protocol defines queries to obtain the list of available physical devices, their characteristics and their locations.

Specific attributes vary with the class of the LDev. For instance, a player may have associated with it volume levels on each of its channels while a listener may have associated with it recording volume levels per channel. A video window may have attributes such as brightness, color levels, tint, etc.

6.1.2. Continuous Media Connections

The X server listens for client connections on a given port. The same is true of a CMEX server, but the CMEX server also listens for connections on another port, called the Continuous Media port. Connections received on the CM port are not treated as client connections, but as CM connections.

Each CM connection can be attached to a given CLDev (see 6.2.3) or LDev. If the type of the CLDev is `output`, any input received by the server on the connection attached to that CLDev is automatically interpreted as output to be rendered on it, provided that the CLDev is mapped. Conversely, if the type of the CLDev is `input`, any output generated by that CLDev will be sent over the connection attached to it, again provided that the CLDev is mapped. This association of connections with CLDevs eliminates the need for CM sources to be X clients, and allows the use of protocols for CM transfer that are better suited to this task than the X protocol.

A connection is a resource in CMEX. The approach of treating connections as separate resources has a very important advantage. Since connections are expensive to set up, it would be useful to reuse connections for different logical devices. If connections can be identified as resources in the server, caching of connections is possible.

6.2. Scenario 2 - Output video and associated audio

In the second example (Figure 6.2), the client wants to output a combination of continuous media streams, say video with stereo audio. The digital information is encoded in an interleaved representation called a **rope**[6]; each of the three constituent parts is called a **strand**. As above, the client first creates a CM connection over which the interleaved information (the ''rope'') is to be sent. In this case the client creates three logical devices (two for audio, one for video). These are combined into a **composite logical device** (CLDev), which is attached to the CM connection. The server is informed of the rope's data format. The CLDev is mapped, causing the constituent LDevs to be mapped to their corresponding physical devices. When data is sent over the CM connection, it is disassembled into its components strands, which are played through the corresponding output devices.

We have introduced three new abstractions: ropes and strands as data representation abstractions, and composite logical devices. We discuss these abstractions in some more detail.

6.2.1. Strands

A *strand* is a single ''track'' of CM data (*i.e.*, a single audio or video track) encoded in a byte stream. A *strand format* is a specification of this encoding. Each LDev has an associated strand format, allowing the server to interpret incoming data for output to a physical device, and to encode data from a physical device onto an output stream.

The set of strand formats can be divided into *classes* where each class corresponds with an algorithmically different encoding/decoding method. Audio strand classes include linear PCM, DPCM, ADPCM, mu-law, and A-law. Video strand classes include px64K, MPEG, DVI SVM (special video mode, 9bpp), and DVI mono. Some classes have additional parameters. For example, linear PCM is parameterized by number of bits per sample, and number of samples per second. DVI SVM is parameterized by image resolution and total average data rate. A strand format is a combination of a class and a set of parameter values.

[6] For example, the combined audio/video in a DVI file would be viewed as a rope.

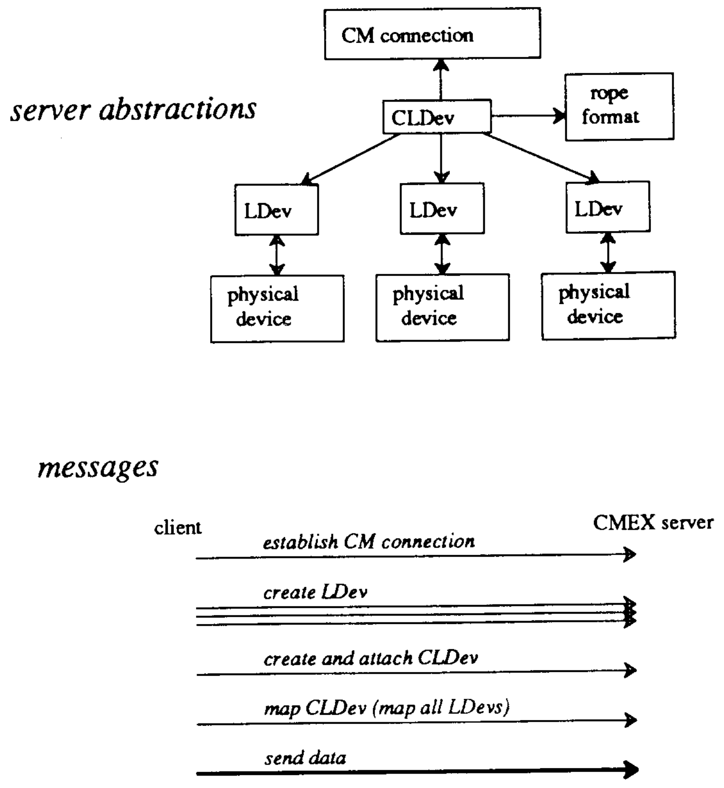


Figure 6.2
The synchronous parallel combination of continuous media streams.

A CMEX server may support any set of strand formats. In some case the server will have to do work in software or DSP (*e.g.*, sample rate conversion) to handler a strand format. The supported strand formats are obtained by the client through queries to the server, which return a list of supported strand formats. The list is ordered by the amount of work needed for conversion; the client should pick a format that is easy for the server, if possible. When a client creates an LDev, it must specify the strand format for that LDev. If the specified strand format is not supported by the server, an error is returned and the LDev creation fails.

6.2.2. Ropes

Several strands of CM data are often associated with each other in a temporally "parallel" manner. That is, the strands are intended to be played in synchrony with each other, whether or not all are intended to be used simultaneously. Examples of parallelism of this kind might include:

- Association of one video strand with one audio strand (*e.g.*, monoaural "TV program" or "movie");
- Association of more than one audio strand with one video strand (*e.g.*, stereo program, or bilingual program);

- Many audio strands associated (*e.g.*, 24 track audio studio master tape);
- Many audio and video strands (*e.g.*, audio/video postproduction clips);
- Many video strands (*e.g.*, different, simultaneous views of the same real world scene, such as views out different windows from the cockpit of a flight simulator).

Strands associated in this manner are usually stored and/or transmitted in a single byte stream; sections of the strands are interleaved onto this stream. This allows several parallel strands to coexist in a single file or, in the case of network transmission, allows parallel strands to be transmitted over the same network connection. It also simplifies synchronization, as the sequential nature of the storage or transmission medium guarantees that the ordering of the interleaved chunks is preserved over storage and retrieval, or transmission, and that the data rates of the individual strands are matched over the long term.

In CMEX, we call this association of strands a *rope*. Different methods may be used to combine and separate the strands (at the transmitting and receiving ends of the connection). A method of combination is called a *rope format*. Examples of rope formats include DVI AVSS format [18] and Sun MMFS format. A rope format is the method is used to combine the strands, and says nothing about the formats of the individual strands. This allows independence between the rope format and the strand formats: A given rope format can be used with different types of strands, and conversely the same set of strands could be cast into different rope formats.

If a multi-strand rope is to be sent to a CMEX server, it is sent to a Composite Logical Device (CLDev) (see 6.2.3). Just as LDevs have strand formats, a CLDev has an associated rope format. This rope format allows the server to properly separate incoming data for output to a set of logical devices, and to properly combine data from a set of logical devices into a rope for output onto an output stream. A CLDev includes a list of the constituent LDevs. When a connection is attached to a CLDev, all data arriving on that connection is separated according to its rope format. Each strand is then sent to the corresponding LDev for output on the appropriate physical device.

6.2.3. Composite Logical Devices (CLDevs)

A composite LDev (CLDev) represents the set of LDevs that is the source or sink of a rope. A CLDev is a resource in CMEX. The generic operations on CLDevs include *create*, *destroy*, *map* and *unmap*, as for LDevs.

There are two classes of CLDevs: `Input` or `Output`. Each CLDev has an ordered list of its constituent LDevs. The create request takes as arguments the class of the CLDev, its resource ID and a list of constituent LDev resource IDs. The individual LDevs that constitute a CLDev have an independent existence. This is necessary in order to manipulate LDev-specific attributes like volumes on audio devices or intensity on video-windows.

Mapping a CLDev is equivalent to atomically mapping the constituent LDevs. Similar semantics hold for the unmap request. An LDev can only be part of a single CLDev.

Each CLDev has an associated set of attributes. The three most important attributes of CLDevs are:

- *Rope*: The format of data to or from the CLDev is indicated by its rope attribute. Each LDev in the CLDev is associated with the corresponding strand in the rope. The rope attribute may be changed.
- *Connection*: The CLDev has associated with it a connection over which data of a particular rope is sent. Each constituent LDev inherits its connection attribute from its CLDev.
- *The logical time system*: Each CLDev is associated with a logical time system. Each constituent LDev inherits the logical time system from its CLDev (see 6.4.1).

6.3. Scenario 3 - Playback from file server

In the third example (see Figure 6.3), the CM data is shipped to the CMEX server from a "third party", in this case a file server.

This illustrates that having a CM connection (see 6.1.2) as a separate abstraction allows the server to send and receive directly from the "third party". This is done to reduce overhead in the client and to reduce overall delay. This approach does not require the third party to speak the CMEX protocol.

6.4. Scenario 4 - Synchronized Playback

In the fourth example (Figure 6.4) several strands, each coming from a different source, are played synchronously. The CMEX server ensures that the strands start playing at the same time and remain in lockstep. This is done by mapping the LDevs to a single **logical time system** (LTS).

6.4.1. Logical Time Systems

As a basis for the timing of continuous media streams, CMEX uses the *logical time system* (LTS) abstraction. An LTS is a temporal coordinate system in which the timing of input and/or output events is expressed. An LTS is first *created* and then *started* by the client. After an LTS is started, it has a monotonically increasing *current value*. The rate of advance of an LTS (relative to real time) is up to the server. The rate may be determined by the clock of an I/O device (*e.g.*, a DAC or ADC) or by the server's real-time clock.

A special server-defined LTS, `LTS0`, always exists. This represents the server's notion of "real time".

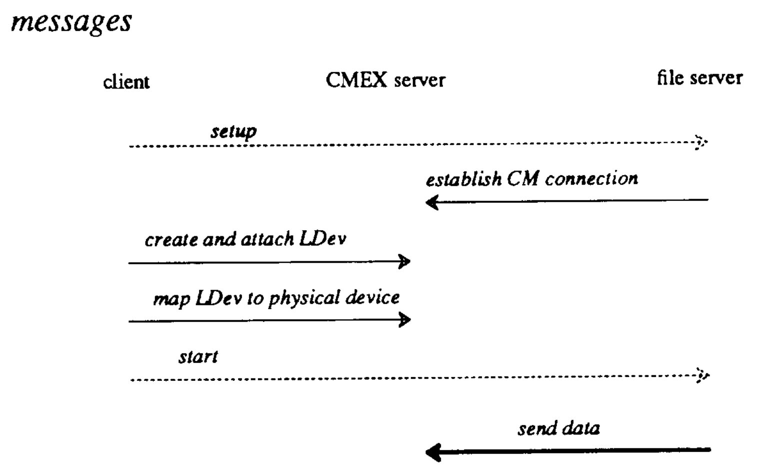


Figure 6.3
Simple playback from a third party.

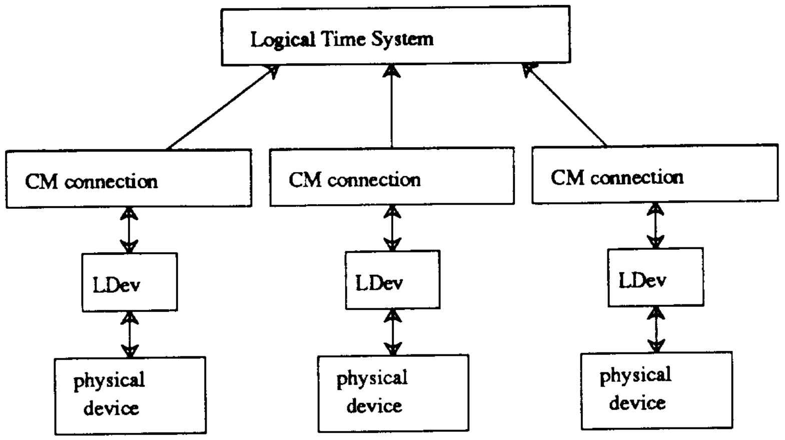


Figure 6.4
Synchronized playbacks of strands from several sources.

Clients can specify that a continuous media stream be synchronized with a specific value of the LTS. Clients can also specify that discrete events (such as graphics output) take place at a specific value of an LTS.

An LTS is a resource in CMEX. It may be *created* and *started*. Every LDev and CLDev has an LTS attribute. The default value for this attribute is `LTS0`. If the LTS associated with an LDev or CLDev is `LTS0`, input from or output to the LDev or CLDev occurs immediately and is not synchronized with I/O from other LDevs or CLDevs.

In some cases, it might be necessary to synchronize two or more continuous media streams. This happens when, for instance, the audio associated with a video stream arrives on a separate connection. This section describes how such synchronization may be achieved using logical time systems.

Each of the LDevs or CLDevs which are to be synchronized are associated with the same LTS. When the server receives a request to start an LTS, it starts displaying output to the LDevs simultaneously. It is upto the clients to maintain sufficient backlog at the server before starting the LTS. The server is not responsible for mapping LDevs and CLDevs which are associated with an LTS, when the LTS is started.

Flow control between server and client is achieved as follows. The server buffers up some unspecified amount of data if the LTS is not started up. However, once the buffer is exceeded, subsequent data is discarded. The client receives an event informing it that the time system has not been started. Further, if after the LTS has been started, clients either fall behind or run ahead of the LTS, flow control messages are sent. These flow control messages are of the form: the server received a message with timestamp x when the logical time on the LTS was y. These events are merely hints to the client, which can ignore them.

6.5. Scenario 5 - Performance Guarantees for Playback

In the final example (Figure 6.5) a single CM stream is played with guaranteed performance. The client establishes a session (using SRP or a similar protocol) after setting up the resources for simple playback. The CMEX server participates in the end-to-end session establishment by reserving the necessary local resources (CPU and DSP).

7. CONCLUSION

We have described the properties of *integrated digital continuous media* (IDCM) as an approach to offering audio and video interfaces in a distributed computer system. The design of an IDCM system poses challenging problems at many levels: device scheduling, network protocols, operating system structure, media-server architecture, storage, application toolkit, and user-interface design. We have suggested approaches to some of these problems, embodied in the following design components:

- The DASH resource model defines a characterization of workload, an abstract interface to hardware resources, and an end-to-end resource reservation algorithm. The model is tailored to continuous media, and can be used to provide the types of end-to-end performance requirements needed by continuous media applications. Furthermore, the resource abstraction can be applied to many existing scheduling policies, so that existing software can be used with minimal modification.
- The Session Reservation Protocol (SRP) implements the DASH resource model in the context of TCP/IP networks. It is used to provide guaranteed performance for traffic on connections of arbitrary upper-level protocols, including TCP. Its design accommodates situations in which not all hosts implement SRP.

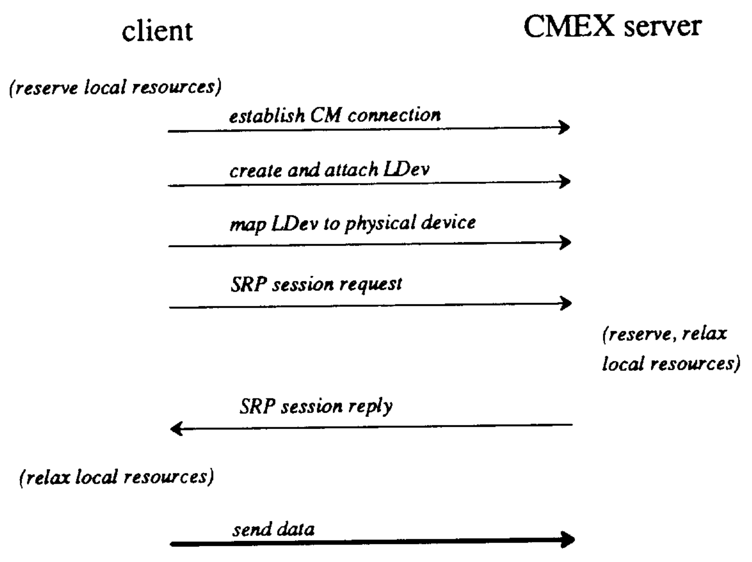


Figure 6.5
Simple playback with guaranteed performance.

- UNIX-type operating systems can be made compatible with the DASH resource model with only limited modification. The schedulers for potential bottleneck resources (CPU, network, disk) must be modified to support sessions, and the ability to pin pages in physical memory must be provided. Other features, such as multiple processes per address space, preemption in the kernel, and VM remapping for data movement, are desirable but not necessary.
- Continuous Media Extensions to X (CMEX) allows digital continuous media to be added as a backwards-compatible extension to a standard window server. Our design allows interleaved data streams, synchronized output from separate streams, and a flexible scheme for distribution of input. It can exploit, but does not depend on, the SRP protocol.

As of the writing of this report (February 1990), the above software components are still mostly in the design stage (an implementation of SRP is in progress). We plan to begin a prototype implementation of CMEX in the next few months. These components form the basis for an IDCM system. Other components will be required to complete this system and make it easy to use. These components include the following:

- Facilities for mass storage of continuous-media data. It may be possible to use existing file systems, such as UNIX/NFS and Andrew, as a basis; it may be necessary or desirable to start from scratch. Existing work in this area includes the Sun multimedia file server project [22] as well as various standards for CD-based storage [10].
- User-interface toolkits for continuous media. Applications will increase in complexity both in their internal structure (*e.g.*, because of the use of multiple concurrent processes to handle CM data streams) and in their interface (because of concurrency and mixed-mode interactions). Toolkits, interface specification languages, or other approaches will be useful in managing this complexity.
- Facilities for multi-user CM applications. Audio/video conferencing and collaboration on CM applications are both important uses of an IDCM system. As described here, our design concentrates on point-to-point communication, which is appropriate for single-user applications and two-person conferencing. Support for multiuser applications can be built using these primitives, but it may be more efficient to extend the low-level abstractions to handle multiuser requirements (multicast, n-way synchronization, etc.).

REFERENCES

1. D. P. Anderson, S. Tzou, R. Wahbe, R. Govindan and M. Andrews, ''Support for Continuous Media in the DASH System'', *ICDCS10*, Paris, May 1990.
2. D. P. Anderson, R. G. Herrtwich and C. Schaefer, ''SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet'', Tech. Rep.-90-006, International Computer Science Institute, Feb. 1990.
3. M. Andrews, ''Guaranteed Performance for Continuous Media in a General Purpose Distributed System'', Masters Thesis, UC Berkeley, Oct. 1989.
4. B. Arons, C. Binding, K. Lantz and C. Schmandt, ''The VOX Audio Server'', *Multimedia '89: 2nd IEEE COMSOC International Multimedia Communications Workshop*, Ottowa, Ontario, April 20-23, 1989.
5. B. N. Bershad, E. D. Lazowska and H. M. Levy, ''PRESTO: A System For Object-Oriented Parallel Programming'', *Software Practices and Experience 8*, 8 (August 1988).
6. W. R. Byrne, T. A. Kilm, B. L. Nelson and M. D. Soneru, ''Broadband ISDN Technology and Architecture'', *IEEE Network*, Jan. 1989, 23-28.
7. S. Cheng, J. A. Stankovic and K. Ramamritham, ''Scheduling Algorithms for Hard Real-Time Systems'', in *Hard Real-Time Systems*, J. A. Stankovic and K. Ramamritham (editor), IEEE Computer Society, 1988, 150-173.
8. R. L. Cruz, ''A Calculus for Network Delay and a Note on Topologies of Interconnection Networks'', Ph.D. Dissertation, Report no. UILU-ENG-87-2246, University of Illinois, July 1987.
9. C. Dellar and R. Calnan, *An Audio-Video Server*, Olivetti Software Technology Laboratory, Oct. 1989.
10. K. A. Frenkel, ''The Next Generation of Interactive Technologies'', *Comm. of the ACM 32*, 7 (July 1989), 872-881.
11. K. Lantz, ''Structured Graphics for Distributed Systems'', *ACM TOG 3*, 1 (Jan. 1984), ??.
12. L. F. Ludwig and D. F. Dunn, ''Laboratory for Emulation and Study of Integrated and Coordinated Media Communication'', *Proc. of ACM SIGCOMM 87*, Stowe, Vermont, Aug. 1987, 283-291.
13. A. C. Luther, *Digital Video in the PC Environment*, McGraw-Hill, 1989.
14. J. McCormack and P. Asente, ''Using the X Toolkit or How to Write a Widget'', *Proceedings of the 1988 Summer USENIX Conference*, San Franscisco, CA, June 20-24, 1988, 1-14.
15. J. Postel, ''Transmission Control Protocol'', *DARPA Internet RFC 793*, Sep. 1981.
16. J. Postel, ''Internet Protocol'', *DARPA Internet RFC 791*, Sep. 1981.
17. R. Rashid, A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. Bolosky and J. Chew, ''Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures'', *IEEE Trans. on Computers*, Aug. 1988, 896-908.
18. G. D. Ripley, ''DVI - A Digital Multimedia Technology'', *Comm. of the ACM 32*, 7 (July 1989), 811-822.
19. F. E. Ross, ''FDDI - A Tutorial'', *IEEE Communications Magazine*, May 1986, 10-15.

20. R. W. Scheifler and J. Gettys, ''The X Window System'', *ACM Transactions on Graphics 5*, 2 (Apr. 1986), 79-109.

21. L. Sha, R. Rajkumar and J. P. Lehoczky, ''Priority Inheritance Protocols: An Approach to Real-Time Synchronization'', Tech. Report CMU, Carnegie-Mellon University, Pittsburgh, PA-CS-87-87-181, Nov. 1987.

22. *Multi-Media File System Overview*, Sun Microsystems, Aug. 1989.

23. D. B. Terry and D. C. Swinehart, ''Managing Stored Voice in the Etherphone System'', *ACM Trans. Computer Systems 6*, 1 (Feb. 1988), 3-27.

24. S. Tzou and D. P. Anderson, ''A Performance Evaluation of the DASH Message-Passing System'', Technical Report No. UCB/CSD 88/452, Computer Science Div., EECS Dpt., Univ. of Calif. at Berkeley, Nov. 1988.

25. L. W. and C. R., ''Digidesign's Sound Accelerator: Lessons Lived and Learned'', *Computer Music Journal 13*, 1 , 36-46.