A Protocol Processing Architecture for Networked Multimedia Computers

R. Gopalakrishnan **Andreas D. Bovopoulos**
Department of Computer Science
and
Computer and Communications Research Center
Washington University, Saint Louis, USA.

Abstract

Multimedia workstation architectures differ from current architectures in these respects — they have multiple specialized processing units, a high speed I/O interconnect mechanism, a high speed broadband network interface and a real-time multitasking operating system (OS) that provides QoS guarantees. These systems will primarily be used to run distributed applications that require high network throughput and predictable delay and delay jitter for real-time traffic. We argue the need for a different protocol organization and processing architecture in order to achieve this.We show how the emerging hardware architecture and OS structures favor a "decentralized protocol processing" approach, that takes advantage of the data delivery mechanism provided by the hardware to improve in-band (protocol data) processing, and the sophisticated OS mechanisms based on communicating objects to improve the out-of band (control) processing. We discuss the need for providing end-to-end QoS guarantees for applications and discuss how it can be naturally incorporated in the proposed architecture.

1. Introduction

Several hardware architectures have been proposed for distributed computing platforms capable of processing multimedia data such as video and audio [2, 9, 11, 13, 16, 19, 20, 22]. These architectures resemble loosely coupled multiprocessors where each processing device is specialized to process one (or more) types of media. To support the high transfer rates associated with multimedia traffic, some of the proposed architectures [2, 9, 20, 22] incorporate an I/O interconnect mechanism that uses fixed size packets, as the basic unit of data

This work was supported by an industrial consortium of Ascom Timeplex, Bellcore, BNR, DEC, Goldstar Information and Communications, Italtel SIT, NEC America, NTT, SynOptics Communications.

Computer & Communications Research Center, Bryan Hall 405 – Campus Box 1115, Washington University, One Brookings Drive, St. Louis MO 63130-4899. e-mail: gopal@wuccrc.wustl.edu, andreas@wuccrc.wustl.edu

transfer within the machine. In addition, the interconnect is able to provide a direct data path between any pair of devices, thereby eliminating the need for CPU intervention and associated data copying. The hardware enhancements are not sufficient in themselves to support multimedia applications, and they need to be accompanied by appropriate modifications to the protocol processing architectures. This is essential in order to be able to satisfy the high throughput, and real-time transport and processing requirements of multimedia applications.

The changes in the hardware architecture will also have a significant impact on the OS. A single monolithic kernel structure is inappropriate for a multiprocessor system. Modern OSs are increasingly being structured around a microkernel [8, 25] that provides low level services, and a collection of configurable service objects that implement higher services. This approach is modular, extensible and can be easily distributed across machine boundaries. A distributed, real-time microkernel based OS can be adapted to meet the needs of the proposed architectures. Therefore protocol modules must be structured to take advantage of the abstractions and services provided by such an environment.

The rest of the paper is structured as follows. Section 2 describes the hardware architectures that have been proposed for multimedia machines, and their influence on the protocol organization and processing architectures. Section 3 considers OS requirements and options for the proposed architectures, and an OS structure is proposed that satisfies these requirements. Section 4 describes a protocol processing architecture that derives from the hardware architecture and OS structure. Section 5 demonstrates how a common application such as videoconferencing can be mapped to this architecture. Section 6 presents the conclusions.

2. Hardware Architecture and Protocol Processing Architectures

It has been realized that it is difficult if not impossible to support real-time data streams on conventional architectures because of large frame sizes (up to several megabytes), need for real-time processing, and their sustained bursts over long durations (up to several minutes). We will look at the shortcomings of current hardware platforms, and how they have been overcome in some of the proposed solutions. The rationale behind the architectural enhancements and their implications for the protocol processing architecture are described.

2.1. Inadequacy of Current Architectures

Current systems suffer from I/O bottlenecks and processing bottlenecks. Most computer systems do not have much emphasis on I/O subsystem. This is because, existing peripherals are slow and do not generate large volumes of data. This is no longer true of multimedia devices, and so the I/O system becomes a performance bottleneck. Furthermore, because multimedia data is not reusable for most applications, existing mechanisms such as caching

cannot alleviate the I/O bottleneck. The processing bottleneck arises because of the master-slave relationship between the CPU and peripheral units. All I/O is interrupt driven and requires CPU intervention. Modern RISC processors with large caches, have a lot of latency and context switch overhead, which can adversely affect performance for real-time streams. Expensive interrupt processing reduces the time available for normal operation, making it all the more difficult to handle the high data rates of multimedia traffic.

Another source of inadequacy is due the excessive emphasis placed on system throughput, which causes exclusive resources such as buses and the CPU, to be overcommitted in order to achieve maximum utilization. This limits the ability of these systems to support devices and services that require guaranteed response times. In the following subsections a few proposed solutions are described.

2.2. I/O Subsystem Enhancements

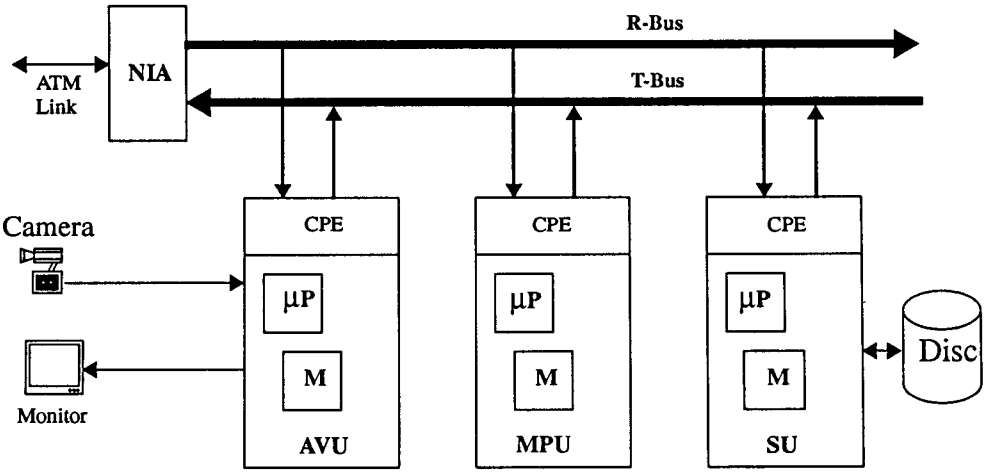
Almost all I/O architectures proposed, seek to extend the ideas developed for high speed networks into the computer. One approach is to retain the bus based backplane as the I/O interconnect, but provide performance guarantees to devices on the bus similar to the way in which the broadband network does for connections. The other approach is to extend the network hardware and data formats into the computer, with devices being the endpoints.

The first scheme is followed in [20] where I/O between devices within the machine are done over *virtual circuits (VC)* in accordance with the transfer rates and delay bounds negotiated with the endpoints of each such VC. Thus several *logical* data paths can be setup over shared physical buses in a flexible manner. To allow concurrent transfers whenever possible, buses may be isolated by switches which reconfigure for inter-bus transfers. Scalability can be achieved by grouping buses in horizontal and vertical directions.

In the second scheme, some designs move the host–network boundary further into the computer, to each peripheral device thereby giving them a direct access path to the network interface. This can be achieved, by preserving the format of the data arriving from (or going to) the network and distributing it to each device. For instance [2] uses a scheduled, reservation based bus to carry ATM cells to each device on the bus (Figure 1). This idea can be carried further by adopting the switching fabrics developed for network switches as the interconnect mechanism, and by using the ATM cell as the unit of data interchange between all devices in the system, such as CPU, memory, and peripherals. Several mechanisms and topologies have been proposed to realize this, and one of them is the *Desk Area Network (DANs)* [9, 24] (Figure 2). The design in [9] uses the *Fairisle* space division switch and the design in [24] is built around the *VuNet* fabric which interconnects CPUs, display, disk, camera, microphone and speaker of a workstation.

2.3. Shift to Multiprocessor Architecture

The processing bottleneck mentioned earlier can be overcome to a large extent by moving towards a multiprocessor architecture [9]. Multimedia applications exhibit a high degree of coarse grain parallelism as far as the processing of the media streams are concerned.



NIA: network interface adapter
MPU: main processing unit
AVU: audio/visual unit
NIA: network interface adapter
R-Bus: receive bus
T-Bus: transmit bus
CPE: cell processing engine

Figure 1: Hardware Organization of SYMPHONY

Furthermore, since the processing hardware as well as the algorithms used for each media are different, it would be beneficial to have multiple specialized units, that partition the application on the basis of the media involved, and process the streams concurrently on their respective units. The coupling between these processors is loose, and the resulting architecture is also highly asymmetric. This parallelism makes it all the more important to allow each processing unit to perform I/O independent of each other and justifies the choice of the I/O subsystem. The use of multiple processors introduces the need to develop new mechanisms to implement temporal synchronization between related streams and will be addressed in a later section.

2.4. Device–Interconnect Interface and Handshaking

A typical multimedia computer would support media such as video and HDTV that require a high performance processor to perform the "in-band" processing. Therefore the burden of per cell processing should not rest with the device processor, but should be relegated to a hardware unit referred to as the Cell Processing Engine (CPE) [2, 24]. The processor at the device interfaces to the CPE in several ways, such as by direct manipulation through special processor instructions [24].

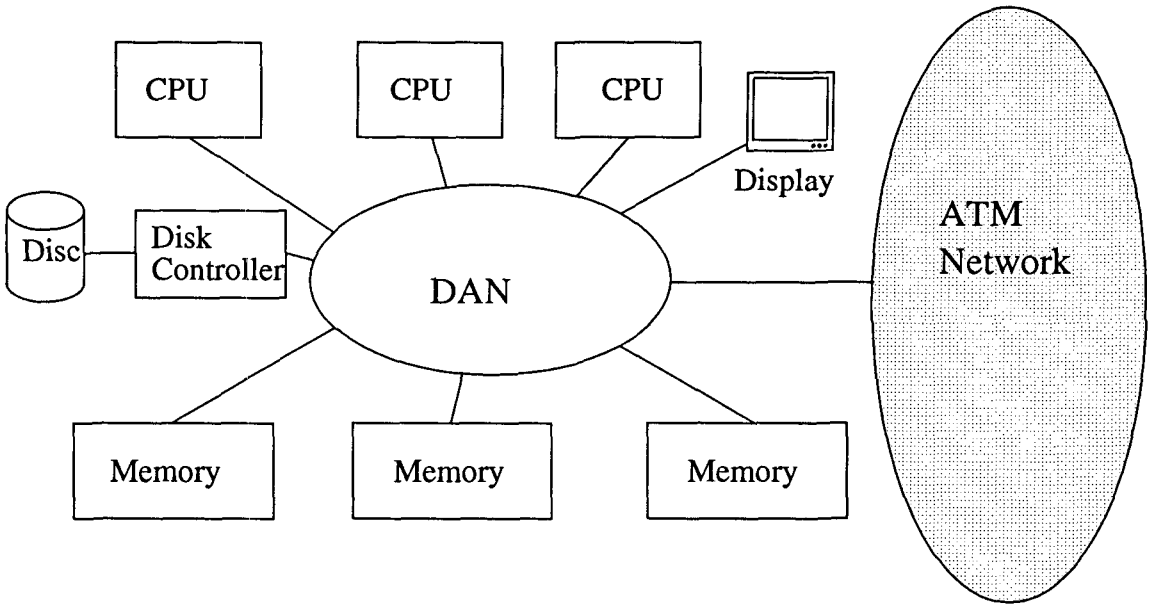


Figure 2: The Desk Area Network Organization

2.4.1. QoS Considerations for I/O Subsystem. An important feature of multi-service broadband networks is that they provide services of varying quality according to application requirements. In order to extend this to the applications, the I/O interconnect must provide predictable and negotiable performance. Here also, solutions for broadband networks can be adapted for use within the machine. One solution is to setup connections over the interconnect between devices and perform resource allocation as per the needs of the connection. This allows us to integrate the QoS requirements within and outside the computer in a uniform manner. Because the number of connections that need to be setup between machine components are small, issues such as admission control are simplified to a large extent. An arbitration mechanism (centralized/distributed) must be implemented to share the interconnect.

Apart from a arbitated way of using the system interconnect, guaranteeing QoS requires that the resource management within each device itself must be negotiable and controlled. Each device must provide a negotiation interface for the use of its resources.

2.5. Implications for Protocol Processing Architecture

The enhancements mentioned above, affect the way in which the flow of data and control occur in different layers of the hardware and software components. The availability of sufficient processing power at each device and the ability of the I/O interconnect to stream data directly to each device favors the strategy of moving protocol processing modules into the data stream path rather than copying data into the address space of the protocol process. This in turn requires more sophisticated control over the processing entities executing on different processors and has implications for choosing efficient interprocess communication

mechanisms over the I/O interconnect. The QoS requirement suggests parameterization of protocol modules in terms of performance within the same protocol family [6].

Section 4 describes a design that takes into account the above mentioned points. Because protocols execute within the OS environment, the nature of the OS support that will be provided on these hardware platforms needs to be looked into. This is the subject of the next section.

3. Operating System Options for Multimedia Machine Architectures

The shift from a uniprocessor to a multiprocessor architecture, and the modifications to the I/O subsystem, necessitate changes to the OS structure. It is important to take these changes into account because they strongly affect protocol processing architecture and its efficiency. The hardware architectures considered exhibit considerable asymmetry because of the partitioning of function as well as data between the processing units. The OS must hide this asymmetry from the application programs. A distributed operating system (DOS) [23] is well suited for this purpose, where each processing component manages its own resources and provides its specialized services through a service interface. The minimum requirements from the OS at each device is that it provides a process abstraction, and a high level interprocess communication (IPC) mechanism, so that applications can access the services of that device in a uniform manner. Depending on the functionality required, other facilities such as file system support may be provided.

A natural way to meet these requirements is to have a *microkernel* [15] on each device that manages local hardware and provides support for processes and IPC (Figure 3), and to implement other system services in the user space. Two important features of the OS must be support for real-time process scheduling, and resource management mechanisms that provide QoS guarantees. The following subsections further elaborate on these areas.

3.1. Microkernel Organization

Real-time microkernels have been proposed as viable solutions to implement distributed environments for cooperative computing [8]. This approach makes it easier to integrate subsystems distributed over various communication media, both inter-system (LAN), or intra-system (DAN). To make the asymmetry transparent to the programmer, the process abstraction as well as the IPC mechanism must be identical for all devices. Finally, the microkernel will need to implement local processor scheduling and memory management.

3.2. Process Abstraction

One of the tasks of the OS is to support the notion of address space (code and data) and process threads (control). An address space could be partitioned over one or more processor memories, and a process could have one (or more) thread(s) that execute on

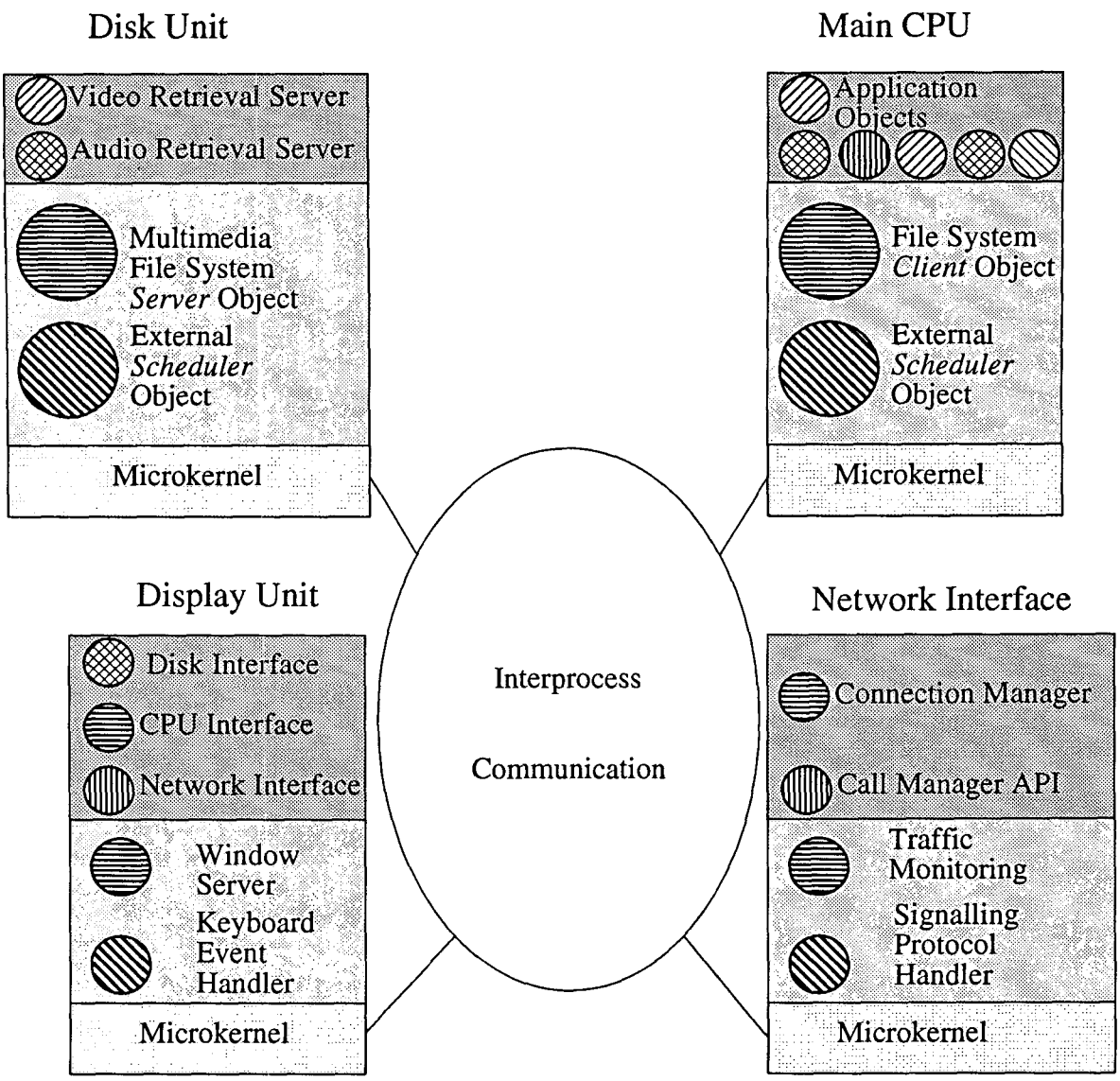


Figure 3: Distributed Real Time Microkernel Organization of the SYMPHONY Multimedia Workstation

each processor. In the case of asymmetric multiprocessor architecture introduced earlier, the low level of data and code sharing does not require the address space to be shared between processes. Instead, a client-server structure is adequate for most applications in which the client and server processes are on separate devices and communicate using the IPC mechanism. Because multimedia applications will be complex, software engineering concerns favor the usage of the object paradigm as a uniform method of structuring static entities (program code) as well as dynamic entities (processes). Regarding processes as objects is more powerful than the client–server paradigm because in the former case, either process object can initiate a method invocation on its counterpart. The view of a process as

an object that provides a method interface will be assumed for the rest of our description.

3.3. Interprocess Communication Abstraction

Message passing is a natural way to build distributed systems. In the case when a microkernel controls each autonomous unit, high level IPC abstractions must be provided so as to hide the distributed nature of the underlying hardware and services. The reliability and high speed of the I/O interconnect, as well as the proximity of the communicating entities can be exploited to efficiently implement higher level abstractions such as *communicating objects*. The benefit of referring to objects rather than procedures (as in RPC) is that they can encapsulate a set of related services, and are supported efficiently by object oriented languages (such as Objective-C).

We choose to implement communication between disjoint address spaces using object method invocation. For example, a client-server connection is implemented using a pair of connection objects in the client and server processes. The `send` and `receive` methods are implemented as methods in both objects. The connection objects implement these methods using a lower level message mechanism provided by the microkernel. The IPC mechanism is sophisticated, in the sense that it can pass objects as arguments to methods. This allows a lot of high level language features to be implemented efficiently.

3.4. Quality of Service Guarantees

The OS must provide predictable and guaranteed service to applications. Real-time, periodic streams must be guaranteed service before their deadlines. Several real-time scheduling schemes have been proposed for multimedia stream scheduling [7, 10]. Most schemes use priority based preemptive scheduling policies for scheduling active, exclusive resources such as processor(s) and communication channel(s). The priority could be static as in rate monotonic (RM) schedulers, or dynamic, such as in earliest deadline first (EDF) schemes. Microkernels separate mechanisms from policies [15] and thus allow external schedulers to implement specific policies [12] suitable to a particular device.

4. A Protocol Architecture for Multimedia Machines

The preceding two sections have established the need to reexamine protocol architectures for emerging computing platforms that handle multimedia data. Certain desirable features that need to be incorporated, in order to take advantage of the enhancements to the hardware and OS mechanisms have also been brought out. In this section, we present a sample architecture that is built around these features. This work is part of our ongoing research into multimedia workstation design [2]. We briefly mention the important features of the protocols implemented on the endsystems as well as their interaction with the network infrastructure. This will clarify the description of protocol architecture. The protocol composition approach that is adopted for implementing protocols, as well as the network

service abstraction that is assumed needs to be explained, in order clarify various aspects of the protocol processing architecture.

4.1. Compositional Approach to Protocol Organization

In several operating systems (notably UNIX and its derivatives) each layer is implemented as a set of procedures that are invoked by layers above and below it. This kind of organization is considered unsuitable for real-time data transport, partly due to its structural and functional rigidity, lack of consideration for QoS, and being mainly intended for connectionless network services.

A uniform way of incorporating diverse transport requirements is to compose protocol stacks out of primitive protocol function objects. This is an approach that is receiving considerable attention [6, 17, 21] for structuring protocols. This approach is modular and hides the details of protocol mechanisms by providing a generic interface to the calling entity. Features of object oriented languages such as dynamic binding, and inheritance ease the development as well as the composition process. Furthermore, the notion of QoS can be parameterized into the protocol interface and appropriate values can be instantiated during runtime. The connection oriented nature of network service provides the protocol modules with additional information such as maximum data rate, which helps in providing predictable and negotiable performance.

4.2. Network Service Abstraction

The architecture is descussed in the context of a *network call model* which is a service abstraction [3, 4, 5] provided by the B-ISDN network. A call binds a set of logical endpoints and enfolds one or more connections. Typically a connection would be used to transport a single media and each connection would have a particular QoS associated with it. Each connection at an endpoint is assigned a unique VCI/VPI value to demultiplex different cell streams.

In our design, a connection has a *device* as well as a *protocol stack* associated with it. The code associated with the protocol stack is executed on the device and in general is media and application dependent. A distinguishing aspect of this architecture is the concurrency obtained by separating the "in-band" data of distinct streams on the basis of their destination device, and implementing control aspects such as session control and QoS control using local IPC between the controlling application object and the controlling object for the connection. This architecture is well suited to a broad range of applications such as video-conferencing, video retrieval etc. Several important issues, such as mechanisms to construct protocol modules and associate them with connections need to be investigated. Mechanisms to synchronize events on two or more connections is necessary for many interactive multimedia applications.

4.3. Protocol Processing Components

The protocol processing architecture of SYMPHONY [2] is described by the functions of its hardware and software components and their interaction. We mention in brief some of the important components and their functions.

- **Cell Processing Engine :** The Cell Processing Engine (CPE) is a hardware unit that resides on each device on the I/O backplane and performs per cell processing. It's implementation depends on the transfer mechanism and data format used on the I/O interconnect. For a DAN based on ATM cell formats for example, it implements the ATM, segmentation and reassembly (SAR) and some simple ATM adaptation layer (AAL) functions in hardware. The CPE's main purpose is to filter cells destined for its device and to forward cells destined for other devices within or outside the machine. Addresses of devices within the machine could either be local device identifiers, or could be VCI/VPI values as in ATM. In the latter case, the network interface device filters out all outgoing cells and hands them over to the network.

- **Network Interface Adaptor :** The network interface adaptor (NIA) is a device that is connected to the network as well as to the I/O subsystem and forwards cells into and out of the machine. The NIA could change the format of the data units depending on what is used by the local I/O subsystem. In the case of a backplane implemented as a time multiplexed bus, the NIA could write multiple bytes depending upon the bus width. It implements the signaling protocol with the User Network Interface (UNI) and is dependent upon the service platform provided by the network. The Network Services Interface is a server that executes on the NIA in order to perform control operations requested by applications that use the service abstraction provided by the network. For instance, if the network supports the notion of a *call*, the NIA will have a *call manager* that manages the calls that are active on the machine. Other functions of the NIA could include traffic policing and shaping.

- **Call/Connection Manager :** This is a server object that is associated with the NIA. It maintains the local state of connections that are active in the machine. Processes that read or write over a connection, perform control operations on their connections by invoking appropriate procedures of the connection manager object. These control operations could include opening/closing a connection, changing read/write privileges, adding/removing an endpoint and so on. The connection manager uses the signaling protocol to interact with the UNI when the control operation requires negotiation with the network. The connection manager operations are "out-of-band" operations and are not used during normal course of data transfer over the connections.

- **Connection Objects :** These entities are used to refer to connections that have endpoints at the host machine. A connection object is created by the connection manager on the NIA when requested by an application object. The main function of the connection object is to encapsulate the state of a connection. Each connection object is associated with a corresponding object which is in the address space of the process

that requests its creation and which forms the connection endpoint. The endpoint performs all operations such as reading, writing and control on the connection, by invoking methods of the connection object in its address space. The object methods in turn, use the local IPC mechanism to communicate with the connection manager, which then performs the operation. Thus the operations on connections that span several machines appears as local IPC to the applications, and thus provides location transparent communication to applications.

- **Protocol Objects :** Protocol object libraries are made up of class definitions that can be linked to an application process and are instantiated as protocol objects during runtime. A single class usually performs a particular function and different mechanisms are implemented in its subclasses [6]. These instantiated objects have methods that allow them to be linked to form a protocol stack. A single message object can thus be subjected to processing at different layers. Typically, for a multimedia device that consumes data in real time (such as a HDTV device), a server object is implemented that manages the resources of that device. For shared devices, this server would provide guaranteed service with a range of quality parameters to connections that represent sources or sinks of data units. An application that intends to use the device would thus request the server object to open a connection with the required QoS parameters. The server would then negotiate with the NIA for a connection and instantiate a protocol stack for processing the data over that connection.

- **Synchronization Server :** The synchronization server is used to enforce temporal relationships between different media streams. The server associates a precedence graph structure with a set of sessions that specifies the temporal order among events that occur within a sessions as well as between one or more sessions. The sessions inform the synchronization server when a particular event is ready and the server orders their occurrence using a token scheme. The receipt of a token allows an event to occur in a particular session. The specification of temporal relationship can be done using timed petrinets [14] or using directed acyclic graphs (DAGs). The videoconferencing application described next shows how the server is setup by an application.

5. A Videoconferencing Application Example

To illustrate the protocol processing architecture, we take the example of a videoconferencing application. This application may involve three media namely — video, audio and text. Participating sites are members of a call which enfolds the connections mentioned above. On one workstation these three media would be carried in three different connections, and these three connections could have endpoints in one (or more) devices. For example, the camera controller would have write privileges for the video connection, and the display controller would have read privileges for the same connection. It is possible that, the camera and display are on different devices within the system.

This application is mapped onto the architecture at various levels. First the partitioning of activity over the devices that are involved in processing the media streams is done. It

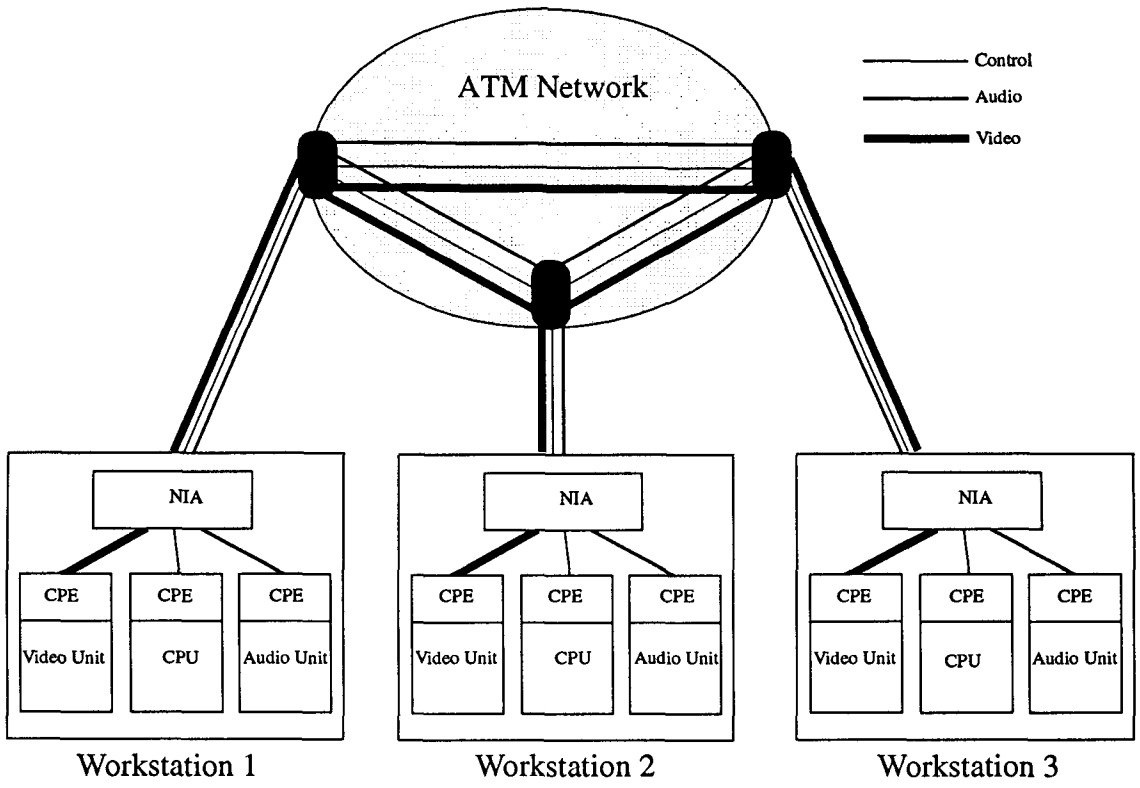


Figure 4: A Videoconferencing Application

is assumed that a server object controls the resources of a device and is also the endpoint of the connection over which the data leaves or enters the device. The control of the session and its "semantics" are handled by a task running on the main CPU. A typical sequence of operations to realize this mapping is given below. The calls shown below would be implemented as part of the Application Programming Interface (API) [6] to network services.

- **join_call**(*call_id*) : This introduces a new participant to the conference session. The *call_id* is an identifier assigned by the transport service provider. The **join_call** function would invoke a method of the **call manager** on the NIA which would return an object identifier. Future operations on the call use this identifier as a handle.

- **join_connection**(*call_id, conn_id, device_id, attr*) : This adds an endpoint to an existing connection (or creates a new one) in the specified call. The *device_id* specifies a device, and a server on that device that will be endpoint of the connection. The *attr* argument specifies connection attributes such as read/write rights and QoS parameters. This call invokes the open method of the server on the specified device which ensures that the local resources are available to guarantee the required service quality. It then contacts the NIA server process to create the connection. The NIA returns a reference to a connection object, to which future operations on the connection are performed. The server on the device now controls the connection and can send or

receive over it. The server would return a session identifier to the controlling task which is used for control operations.

- **bind**(*conn_id, proto_id, params*) : This is used to associate a connection with a particular protocol stack that is referenced by its identifier. The protocol stack would have been composed out of modules instantiated from the object library. The parameters *params* are used to guide the instantiation process.

- **temporal_spec**(*session_id_set, precedence_graph*) : Once a set of sessions have been setup, the synchronization server is notified of the temporal relationships between the streams. The server then operates as described earlier on receipt of event notifications from the sessions in the given set. The precedence graph is the representation of the temporal relations.

The API functions listed above are at a fairly high level and are supported by lower level primitives. A uniform aspect of the application structure is the client server model built on top of an object invocation mechanism. The architecture is flexible to accommodate other applications such as video editing and retrieval.

6. Conclusions

In this paper we have attempted to put into perspective, several multimedia workstation design efforts. We have presented the rationale that these designs share in common. We conclude that fundamental design changes are needed in the area of hardware architecture, OS structuring and support, as well as the protocol organization and processing architectures. In hardware, the trend is towards providing a high speed I/O interconnect (DAN), an asymmetric multiprocessor structure, and integrating the notion of QoS into device management. We justified the need to reexamine current protocol processing methodologies in order to take advantage of the features of these new system architectures. From an examination of developments in operating system support, we conclude that there is a shift towards microkernel based operating systems, that support real-time processing and are designed for distributed operation. We described our approach of having a collection of real-time microkernels to provide a distributed processing environment, within a machine that is based on the hardware architectures described earlier. We present our approach for protocol organization and processing in such an environment and describe some of its important hardware and software components. We believe that this approach maps well to many common applications, and is flexible enough to accommodate systems that share the features described earlier.

References

[1] P.C. Bates and M.E. Segal, "Touring Machine: A Video Telecommunications Software Testbed," Proceedings of the *First International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1990.

[2] A.D. Bovopoulos, R. Gopalakrishnan and S. Hosseini, "SYMPHONY: A Hardware, Operating System and Protocol Processing Architecture For Distributed Multimedia Applications," Technical Report WUCS-93-06, Department of Computer Science, Washington University, 1993.

[3] R. Bubenick, M.E. Gaddis and J.D. DeHart, "Communicating with Virtual Paths and Virtual Channels," *Proceedings of the IEEE INFOCOM'92 Conference,* May 6-8, 1992, Florence, Italy.

[4] J.D. DeHart, M.E. Gaddis and R. Bubenik, "Connection Management Access Protocol (CMAP) Specification," Technical Report WUCS-92-01, Department of Computer Science, Washington University, February 1992.

[5] M.E. Gaddis, R. Bubenick and J.D. DeHart, "A Call Model for Multipoint Communication in Switched Networks," *Proceedings of ICC'92,* 1992.

[6] R. Gopalakrishnan and A.D. Bovopoulos, "Design of a Multimedia Applications Development System," Technical Report WUCS-92-27, Department of Computer Science, Washington University, 1992.

[7] R. Govindan and D.P. Anderson, "Scheduling And IPC Mechanisms for Continuous Media," *13th ACM Symposium on Operating Systems Principles,* 1991.

[8] M. Guillemont, "Microkernel Design Yields Real Time in a Distributed Environment," Technical Report CS/TR-90-65.1, Chorus Systems, 1991.

[9] M. Hayter and D. McAuley, "The Desk Area Network" *ACM Operating Sytems Review,* October 1991.

[10] R.G. Herrtwich, "An Introduction to Real-Time Scheduling," Technical Report TR-90-035, International Computer Science Institute, Berkeley, California, July 1990.

[11] A. Hopper, "Pandora – An Experimental System for Multimedia Applications," *ACM Operating Systems Review,* 1990.

[12] W. Kalfa, "Proposal of an External Processor Scheduling in Micro-Kernel based Operating Systems," Technical Report TR-92-028, Dept. of EECS, University of California, Berkeley, May, 1992.

[13] H.P. Katseff, R.D. Gaglinello, T.B. London, B.S. Robinson and D.B. Swicker, "An Overview of the Liaison Network Multimedia Workstation," *First International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1990.

[14] T.D.C. Little, and A. Ghafoor, "Multimedia Synchronization Protocols," *IEEE Journal on Selected Areas in Communications,* December 1991.

[15] K. Loepere, *Mach 3 Kernel Principles,* Open Software Foundation, January 1992.

[16] P. Nomtahan and R. Kamel, "PX Connection Architecture," *First International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1990.

[17] S.W. O'Malley, L.L. Peterson, "A Dynamic Network Architecture," Technical Report, Departement of Computer Science, University of Arizona, Tucson, 1992.

[18] Martin de Prycker, *Asynchronous Transfer Mode Solution for Broadband ISDN,* Ellis Horwood, London, 1991.

[19] W.D. Richard, J.R. Cox, B. Gottlieb and K. Krieger, "The Washington University Multimedia System," Technical Report WUCS-93-01, Washington University, January 1993.

[20] A. Sah, D.C. Verma and V.G. Oklobdjiza, "A Study of I/O Architecture for High Performance Next Generation Computers," Technical Report TR-91-008, ICSI, Berkeley, January 1991.

[21] D.C. Schmidt, D.F. Box and T. Suda, "ADAPTIVE A Flexible and Adaptive Transport System Architecture to Support Multimedia Applications on High-Speed Networks," Technical Report 92-46, Department of Information and Computer Science, University of California, Irvine, 1992.

[22] "*SPARCcenter*© 2000 Architecture and Implementation," Technical White Paper-Draft Edition of 11/18/92, Sun Microsystems, Inc., 2550 Garcia Ave., Mountain View, CA 94043, U.S.A.

[23] A.S. Tanenbaum, *Modern Operating Systems,* Prentice Hall, 1992.

[24] D. Tennenhouse, M. Ciholas, and J. Davin, "Telemedia, Networks and Systems," Group Annual Report, AR-001, Massachusetts Institute of Technology, July 1991–June 1992.

[25] H. Tokuda, C.W. Mercer, "ARTS: A Distributed Real-Time Kernel, *ACM Operating Systems Review,* July 1989.