Technical Report

UCAM-CL-TR-191
ISSN 1476-2986

Number 191

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

An architecture for real-time multimedia communications systems

Cosmos Nicolaou

February 1990

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

An Architecture for Real-Time Multimedia Communication Systems†

Cosmos Nicolaou
February 8, 1990
Cambridge University, Computer Laboratory

ABSTRACT

An architecture for real-time multimedia communication systems is presented. A multimedia communication system includes both the communication protocols used to transport the real-time data and also the distributed computing system (DCS) within which any applications using these protocols must execute. The architecture presented attempts to integrate these communications protocols with the DCS in a smooth fashion in order to ease the writing of multimedia applications. Two issues are identified as being essential to the success of this integration: namely the synchronisation of related real-time data streams, and the management of heterogeneous multimedia hardware. The synchronisation problem is tackled by defining explicit synchronisation properties at the presentation level and by providing control and synchronisation operations within the DCS which operate in terms of these properties. The heterogeneity problems are addressed by separating the data transport semantics (protocols themselves) from the control semantics (protocol interfaces). The control semantics are implemented using a distributed, typed interface, scheme within the DCS (i.e. above the presentation layer), whilst the protocols themselves are implemented within the communication subsystem. The interface between the DCS and communications subsystem is referred to as the Orchestration interface and can be considered to lie in the presentation and session layers.

A conforming prototype implementation is currently under construction.

---

† To appear in IEEE JSAC on Multimedia Communications.

1. Introduction

A very brief survey of recent work in multimedia communication is presented.

- Work on the real-time transport of voice and video over digital networks.[Ades1986, Lazar1985] Some work has also been done on extending the OSI reference model to cope with multimedia communication.[Lazar1986]

- Work on Multi-Service Networks (MSN) and their associated protocols which are designed with the explicit goal of carrying multiple types of traffic, in particular voice and video, in addition to data. So-called Asynchronous Transmission Networks (ATM)† are the prime candidates for the practical implementation of such networks.

- Multimedia document preparation, presentation and asynchronous (i.e. electronic mail) transport. The media used have primarily been text, graphics, images and voice.[Thomas1985, Christodoulakis1986, Postel1988, Naffah1986, Nicholson1985, Poggio1985]

- Control of PABX functions from a computer system. These systems allow application programs to be written which control and customise the behaviour of the PABX in question.[Root1986, Herman1987, Redman1987]

- The integration of voice communication into a digital network and distributed computing system. These systems allow for the implementation of software PABXs, voice editing and storage, and multimedia (text and voice) document preparation, as well as the real-time transport of voice over a digital network.[Want1988, Calnan1987, Swinehart1983, Swinehart1987]

- The integration of video into a digital network environment. Magnet in particular has concentrated on the architecture of a high speed integrated local area network capable of transporting real-time voice

† ATM networks are commonly agreed to have three principal characteristics: a fixed cell size, asynchronous access and bounded access time.

and video, and on the design and implementation of a special purpose workstation to handle the presentation of these media.[Lazar1987]

Given that the demands made on the DCS by voice are modest compared to those made by video communication, it is not surprising that the greatest level of integration achieved in the above systems is in the areas of voice communication over a digital network and on the control of intelligent PABXs. However as network[Temple1984, Ross1986, Newman1988, Newman1989] and CPU capacity increases it is becoming possible to handle video as effectively as voice. These higher capacity networks and CPUs will be able to support multiple voice and video streams simultaneously, thus allowing for more complex communication patterns than single media point-to-point (e.g. phone conversation) communication, as has previously been the case. It will also no longer be necessary to build entire workstations specifically to handle voice and video efficiently, thus leading to a desire for open systems. An open system is one which can be incrementally extended by the addition of new functionality without disturbing the existing system components.

To summarise, real-time voice and video will be able to coexist within the same system, and if past experience with voice is an accurate guide then there will be a strong desire to integrate voice and video communication into the DCS. The existence of multiple simultaneous data streams gives rise to the need for some means of controlling and synchronising these multiple streams in order to bring about some meaningful communication, whilst the drive towards open systems carries with it the requirement to effectively manage heterogeneity. The architecture presented here directly addresses the issues of synchronisation and heterogeneity.

The rest of this paper is structured as follows. A survey of the requirements of real-time multimedia communication and of the distributed computing system is presented along with a discussion of the interrelationships of these two sets of requirements. The issues of synchronisation and heterogeneity are then discussed in detail, followed by a description the architecture itself and its relationship to the OSI reference model.

2. Real-Time Multimedia Communication Requirements

Real-time voice and video data streams are isochronous in nature, that is, they can be thought of as a stream of finite sized samples which are generated, transmitted and received at fixed time intervals, imposing a set of timing constraints which must never be exceeded. The delay between the generation of successive samples at the stream's source introduces a sampling delay; there is also a transmission delay which refers to the delay between the generation of a sample and the presentation of the same sample at the stream's sink. It is important to realise that the transmission delay must be end-to-end, that is, this delay must be measured from the point at which the sample is generated to the point at which it is presented to the user. The transmission delay consists of the packetisation delay, the network transmission delay and the presentation delay. The packetisation delay is the time taken to generate a sample and transfer it to the network, the network transmission delay is the time taken to transmit the sample over the network, and the presentation delay is the time spent buffering the sample before presenting it to the user. The packetisation delay is probably dominant (in a Local Area Network environment) and the choice of sample size dictates the magnitude of this delay, therefore the sample size must be chosen so as to give an acceptable packetisation delay and also to give acceptable network utilisation. Therefore the sample size will also be influenced by the protocol data unit size of the network protocol used to transport it.

If the source, network and sink ran completely synchronously, without errors, and introduced no queuing delay then the source and sink would always remain in synchronisation and there would be no need for buffering at the sink. Unfortunately there are statistical queuing delays introduced at the source and sink and errors (i.e. lost or corrupted packets) introduced by the network. These delay variations are often referred to as jitter and the sink must implement some buffering scheme to smooth out these delay variations before presenting the samples to the next level up in the protocol stack. There is an additional source of jitter which is due to the clocks at the source and sink running at different rates; any buffering scheme must take account of this clock variation. Samples arriving late, i.e. an excessive amount of jitter (greater than the maximum allowable delay for the stream in question) are treated as network errors; a late packet is about as useful as a lost packet. For this same reason there is no point in using acknowledgement packets to detect lost packets at the source, since a retransmitted packet following a timeout on an acknowledgement will almost certainly arrive late and therefore is as good as lost! The contentious issue is the error rate which can be tolerated before a noticeable degradation in quality occurs. For voice an error

rate of 1%, provided each error burst is shorter than 4ms is often quoted as acceptable, while for video the acceptable error rate is entirely dependent on the coding and compression algorithms used. The Island voice protocol[Ades1986] implements such a buffering scheme for a real-time voice stream; Magnet uses a buffering scheme which is heavily influenced by the coding scheme used for video.[Lazar1987]

For any stream, with a given sample size and a bounded jitter value it is possible to implement a buffering scheme which smooths out the jitter introduced by queuing delays. Given that ATM networks have bounded jitter characteristics the assumption that a bounded jitter parameter is available for a given data path is a reasonable one. Detecting varying clock rates can also be implemented provided a reliable clock is available against which to compare the rate of incoming packets. However it is unlikely that the clock rates will vary by any noticeable amount given the extreme accuracy of modern quartz oscillators.

The requirements for a given stream can be represented as set of properties usually referred to as a QOS (Quality of Service) parameter. The QOS can be used to set up the buffering scheme as required for this stream and also to distinguish the differing requirements of this real-time stream from the requirements of other non real-time connections. This usually takes the form of prioritising packets for the real-time stream in order to minimise queuing delays and therefore jitter, and also using a light weight protocol which does not use acknowledgement and retransmission techniques. Great care must be taken in the implementation of the communication subsystems in order to avoid inadvertently introducing jitter due to the subtle interactions of buffer management, layering and scheduling operations. A strong case is made in[Tennenhouse1989] for avoiding unnecessary multiplexing in a layered protocol stack since this can introduce unacceptable amounts of jitter. The real-time message facility in the DASH operating system[Anderson1988] takes essentially this QOS approach to precisely tailor the behaviour of the communications subsystem to the requirements of the user application.

The DCS at the source and sink of a real-time stream must be able to meet the delay and jitter demands made of it, this implies that a real-time operating system and real-time run time system for applications running over that operating system be used.

2.1 Interrelationship of DCS and Communications Subsystem

The partitioning of functionality between the DCS and communications subsystem must be such that the communications subsystem has sufficient information on the applications communication requirements to efficiently provide them. The application must have sufficient control over and information on the streams provided to effectively control and manage their synchronisation. An elegant solution exists whereby the communications subsystem is informed of the applications requirements via a QOS parameter, whose properties indicate the nature and requirements of the real-time stream and also the details of when and how the synchronisation information required by the application is to be presented to it. The communication subsystem is then able to implement fine level synchronisation using internal buffering and flow control mechanisms, whilst at the same time supplying the application with the information it requires, (in the format and at the time it is required) to implement synchronisation of both individual and separate but related streams at the (probably coarser) granularity with which the controlling application is best able to manage.

3. Distributed Computing System Requirements

A typical DCS will provide a rich set of facilities for implementing distributed applications; including remote procedure call, light-weight thread and synchronisation primitives, distributed naming, type checked languages etc. If an effective level of integration is to be achieved these facilities must be applicable to the multimedia communication subsystem. Therefore the interface provided by the communication subsystem (i.e. the presentation and session level interfaces) must allow for the efficient implementation of these facilities, in particular light weight threads and their associated synchronisation primitives must be efficiently implemented.

If the DCS is to be used to implement control and synchronisation of real-time streams then it must have sufficient information on which to base its decisions. In particular an application executing within the DCS must have sufficient information to determine if related streams are synchronised, and if not, to take corrective action. Ideally the making and acting on of these decisions should be completely integrated with the run-time system. Also any control operations applied to the real-time streams must be synchronised to them, implying that the streams must be structured so as to allow for this synchronisation.

A suitable set of control and synchronisation primitives need to be defined which do not introduce unacceptable amounts of jitter, yet provide a concise and powerful programming abstraction.

4. Presentation Level Synchronisation

It is useful to examine the likely uses of synchronisation in order to more fully understand the nature of the synchronisation decisions and the ensuing actions required.

4.1 Lip-synching

Lip-synching refers to the synchronisation of spoken voice with the movement of the speaker's lips. This synchronisation can be (as it is for film and domestic VCR recordings) achieved mechanically by recording the voice and video on the same physical medium and then using truly concurrent and separate play back equipment for voice and video. For real-time transmission completely synchronous channels may be used for voice and video, as used for television broadcasts for instance. Neither of these approaches is feasible for computer communication over a digital network, since the network and DCS will inevitably introduce some jitter.

It is possible to multiplex the voice and video samples over a single session layer association, however this approach has several disadvantages. The primary disadvantage is that even though voice and video have very different characteristics (and therefore QOS properties) they must be transmitted over the same lower level association with a single QOS parameter. This multiplexing onto a single lower level association leads to inefficiencies resulting from the inability to make use of stream specific information. In particular the job of reducing jitter is very much harder for two independent streams multiplexed onto a single association than if these streams were kept separate. A secondary problem is that the complexity of the source and sink will be considerably increased if, as is highly likely, different, possibly variable bandwidth, codings are used for the voice and video components of the same multiplexed stream. Finally, this scheme dictates that the voice and video originate from a single point.

An alternative is to use separate session layer associations for the voice and video streams, this scheme allows for separate voice and video sinks, but does require some means for maintaining the synchronisation of these related streams. Given that bounded jitter is achievable it is possible to construct a buffering

scheme which maintains the synchronisation of the individual streams over relatively short (minutes) periods of time. It is then left to the application to ensure that these streams remain synchronised with respect to each other over longer periods of time. Two pieces of information are required to implement this synchronisation:

- The rate of change of the jitter per sample over the last n samples.
- The jitter for the current or most recent sample.

The first value enables the application to detect if the stream is losing synchronisation and to take appropriate corrective action, the second provides some positive feedback enabling the application to determine if its actions are having any effect. Corrective action can take the form of modifying the QOS properties for the stream in question, requiring that the communication subsystem allow these properties to be dynamically changed.

It is also possible to use this information to determine if the source and sink clocks are running at the same rate. However a common clock, which is known to be correct, is required to determine that the source and sink clocks are running at the correct, rather than the same, rate.

4.2 User Interface Management Systems and Positive Feedback

There is a strong drive within the User Interface Management System (UIMS) research community towards more concurrent user interfaces and UIMSs which support this concurrency.[Lantz1987] This drive is motivated by the general belief that concurrent input is a natural way for users to interact with computers,[Buxton1986, Buxton1985, Hill1986] and by the desire to build direct manipulation interfaces as described by Schneiderman.[Schneiderman1983] Direct manipulation interfaces are characterised by concurrent input and the provision of timely positive feedback in response to user actions. Hudson[Hudson1987] and Tanner[Tanner1987] explore in detail the demands made on a UIMS by this type of user interface. An important requirement is that the feedback provided should appear instantaneous to the user, thus imposing a maximum response time in the order of 10-40ms (human perception threshold time).

A central aim of multimedia communication is to allow a single user to use a computer as a tool for communication with several other, physically distant, users. This means that the next generation of UIMS which will implement user interfaces to such multimedia communication must extend their view of human computer interaction beyond the current situation of a single user interacting with a single computer. Therefore the UIMS must cope with multiple sources of human input and the very much larger class of errors introduced by the presence of a network and distribution. These errors include communication errors due to the network itself and partial system failures which occur when part, but not all, of the distributed application managing the communication fails.

If a direct manipulation user interface is to be implemented then feedback must be provided not only in response to the local users actions, but also in response to remote users actions and in response to errors. The error feedback generated must reflect the error in some meaningful fashion to the user, thus avoiding the situation where a user is left to stumble across the error in the normal course of his or her communication.

As a simple example consider the situation where a user is running the X window system. This user has a terminal connection to a remote machine, if the remote machine crashes no feedback is given, rather the user is left to determine that the remote machine crashed based on it's lack of response. This is largely a result of the fact that the communications protocol used does not generate any indication that it is having difficulty communicating with the remote machine. This may not in itself seem a great hardship for the user, however if more complex conferencing applications which support communication with multiple users using multiple media are to be built, then the provision of positive feedback becomes essential.

It is useful to think of these errors as synchronisation points, since every time such an error occurs synchronisation is lost and some action must be taken to resynchronise or to abandon communication in some graceful manner. It is also useful to consider all exceptional, though not necessarily erroneous, events as synchronisation points. For instance opening or closing a real-time connection may generate synchronisation events when the first and last (respectively) samples are received. This allows for related streams to be synchronised with respect to each other whenever such a synchronisation point is reached. In the lip-synching example the controlling application may wish to wait until both streams reach their last sample synchronisation point before tidying up the screen display. Similarly if one of the streams stops due

to some error then the application may wish to stop the related stream and display some meaningful message, or take some action to re-start the stream. These synchronisation points essentially define the points at which the controlling application should consider taking some action, i.e. they are events and actions which warrant some form of response.

4.3 A Synchronisation Scheme

This section presents a scheme for implementing the three types of essential synchronisation which have been identified by the previous examples. The first, referred to as isochronous synchronisation is concerned with maintaining the real-time synchronisation of related streams. The second and third deal with synchronisation after some error and after some well defined point has been reached; both of these can be considered as exceptional and as warranting some form of feedback. Note that the loss of isochronous synchronisation is itself an exceptional event requiring some resynchronisation action.

Real-time multimedia streams are considered as having a two-level structure. At the lowest level such a stream is considered to be an ordered sequence of variable, but finite, size samples which are expected to be generated, transmitted and presented at fixed time intervals, i.e. they are isochronous. These samples are referred to as physical synchronisation frames (PSF). At the next level the stream is structured as an ordered sequence of logical synchronisation frames (LSF), each of which consists of a number of physical synchronisation frames. PSFs are intended as the unit of synchronisation within the communications subsystem, whereas LSFs are the unit of synchronisation for the controlling application. It is possible to have a one-to-one relationship between PSF and LSF; the level of indirection provided by this two level structure allows the application to specify the synchronisation granularity which it can best handle. The actual values used will be highly specific to the application, DCS, communications subsystem and network being used, with the restriction that the source and sink within the communication subsystem use the same PSF. Ideally the LSF should be a QOS property, thus allowing the application to specify the unit of synchronisation it requires in a convenient manner.

Figure 1 illustrates how this synchronisation scheme would work for a video stream which the application wishes to control at a video frame by video frame level, whilst the video stream is implemented using a protocol which transmits four samples per frame. The upward arrows indicate synchronisation points at both levels in the stream, though only the synchronisation points occurring at LSF boundaries are

communicated to the controlling application.

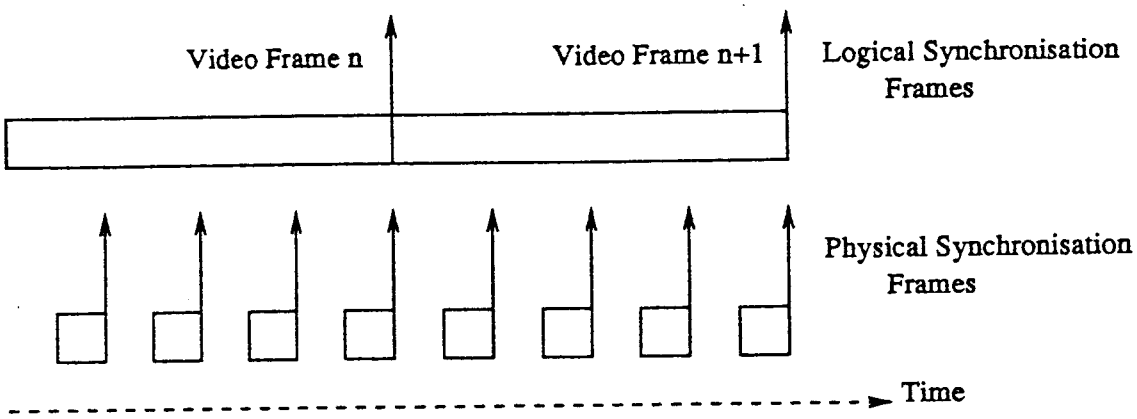

Figure 1. Two-Level Synchronisation

Given this synchronisation scheme, it is then necessary to define how the synchronisation points are indicated to the controlling application, how operations on these streams are synchronised with respect to these streams, and how synchronisation information gathered by the communications subsystem is presented to the controlling application.

The mechanism used to indicate synchronisation points represents an asynchronous flow of information upwards from the communication subsystem to the application, the mechanism chosen must be sufficiently efficient so as not to deter application writers from using it. There are two primary candidates for this asynchronous communication, namely upcalls and event queues: the upcall mechanism is to be preferred since it can be easily used to implement an event queue system, whilst the converse is not true.

Each stream will have an associated set of stream specific operations. These operations must be synchronised with respect to the streams to which they apply, this synchronisation is defined in terms of the streams LSFs. In particular operations only take effect at LSF boundaries, and may be delayed up to some maximum number of such boundaries. These operations are implicitly timed, that is if an operation doesn't take effect within the stated number of LSFs then the communications subsystem must report a timeout error. The benefit of specifying this synchronisation relationship is that both the application writer and the stream implementor have a precise synchronisation model within which to work, thus eliminating potential confusion.

The following figure shows how a "stop" operation would be synchronised with respect to the video stream used in the previous figure. The completion of the operation can be indicated in either of two ways: the operation in question is blocked until completion, alternatively the operation may return immediately (i.e. is not blocked) and an explicit synchronisation event will be generated on its completion. Both blocking and non-blocking modes are illustrated below.

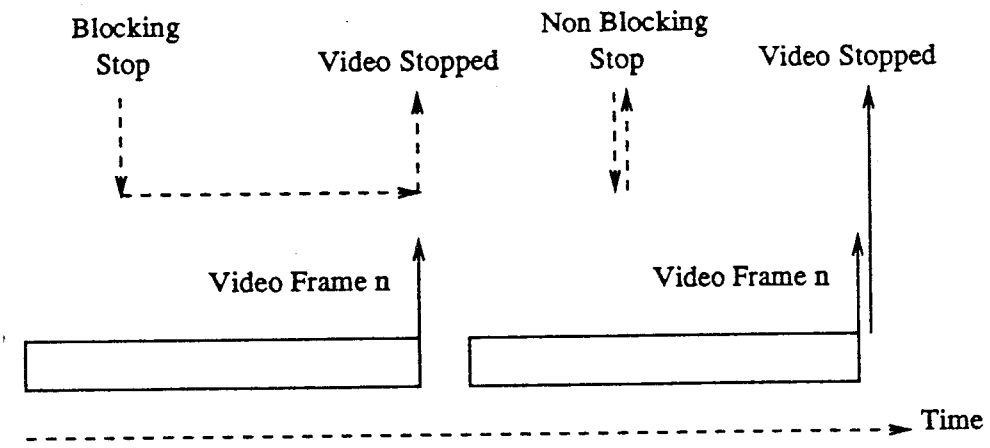

Figure 2. Blocking and Non-Blocking Control Operations

The synchronisation information gathered by the communication subsystem needs to be made available to the controlling application in two distinct situations.

- At regular intervals to monitor the current state.
- At irregular intervals, usually in response to some other synchronisation event.

The first situation can be dealt with by defining these regular intervals as synchronisation points and passing the data as arguments to an upcalled procedure. The second situation is best catered for by a procedural interface, since the application may wish to access this information from within an active upcall.

4.4 Synchronisation Summary and Complete Example

We now have a two-level synchronisation scheme with upcalls occurring at synchronisation points in the upper of the two levels. A procedural interface is provided for setting and modifying QOS properties and for obtaining synchronisation information (this information can also be obtained by an upcall).

Consider a video editor which allows the user to play back stored video in real-time as well fast and slow rates, and also provides a cut-and-paste facility for editing voice and video segments. In order to implement cut-and-paste some means of delimiting the segment to be cut and a means of indicating the location to paste to are required. A reasonable approach is to accompany the play back of video with some kind of "time line" or "scroll bar" as a visible cue on the current position in the video segment. This time-line needs to be updated in time with the associated video. The QOS properties can be used to define an isochronous upcall at some rate, this upcall can monitor the synchronisation of the stream being played and update the time line. If the stream is found to be losing synchronisation then this upcall can take corrective action; this may involve modifying the QOS properties or providing some feedback to the user (e.g. making the time-line flash). If the video is played back at a different rate, then since the synchronisation points are defined in LSFs, the upcall will be called at the new rate, and the time-line will be automatically updated at the new rate. If an error occurs on the play back stream then a separate upcall will be made which can stop the time-line and inform the user of the error. The following section suggests how this scheme can be integrated into the DCS.

4.5 Integrating Stream Synchronisation into the DCS

Concurrent activities typically require synchronisation points, each such point is represented as a synchronisation variable. Such a variable, if set, indicates that the synchronisation point has been reached, if unset then this point has not been reached. Synchronisation variables (SV) are stream specific and are represented as a triple (stream, synchronisation point, value). Three primitives are defined which operate on these variables.

- `WaitForSV( SVexpr, TimeOut )` wait for the expression to become true.
- `SignalSV( SV )` set the synchronisation variable specified.
- `TestSV( SV )` return the value of the synchronisation variable specified.

`SV` is a single synchronisation variable, `SVexpr` is an expression involving any number of SVs separated by one of the following operands:

- `SVAND` boolean and of the operand SVs.

- `SVOR` boolean inclusive or of the operand SVs.

The `WaitForSV` will return an indication of which SVs were set and thus caused it to return, a timeout can also be specified which if exceeded will cause `WaitForSV` to return with a timeout indication. `WaitForSV` will typically be called by controlling threads within the application, whilst `SignalSV` will be called from within an upcalled procedure. If an event queue model is implemented, then the central loop waiting on the queue will call `SignalSV` in response to receiving the associated events.

The DCS may provide some automatic means of generating a large amount of the interface code required to implement this scheme based on information given in the specification of a stream (see below). The design of the run-time system must be such so as not to introduce unacceptable amounts of jitter, therefore the run-time scheduler may wish to base its scheduling decisions on the QOS properties whose synchronisation variables are being signalled; the scheduling of upcalls should be similarly influenced.

5. Managing Heterogeneity

The desire for open systems carries with it a requirement to effectively manage heterogeneity. This problem is particularly acute for multimedia systems in which there may exist many different pieces of hardware capable of performing the same function. The application writer needs to access the functionality provided by the hardware without being overly burdened with the details and differences of the particular pieces of hardware being used. For instance a program to manage a voice conversation using a specially built digital phone should not need to be modified to work with a software phone implemented using a microphone and speaker.

The approach taken treats any piece of hardware or software which can generate real-time multimedia streams as a device. Each such device has a strongly typed interface and an associated implementation. There is no checking to ensure that the interface and implementation are consistent with respect to each other. The interfaces are written in a Device Specification Language (DSL), in which an interface has two components: a stream component which specifies the real-time streams and an operations component which specifies the operations on these streams. The stream components represent the protocols which this device

can use, whilst the operations component represents the control interface to the device and its streams.

The interaction between interfaces is based on the client/server model. A server exports the interfaces it supports and a client must import a previously exported interface in order to use it. An interface is location specific, thus an instance of a server at a given location exports an interface and a client attempts to import an interface exported by a particular location. In a multimedia system which is used to implement user communication, location transparency is of little use, since a person, unlike a replicated software server, cannot be in two places at once. Therefore import requests may specify a particular location, usually this will be the current location of the user(s). Note that if location transparency is required a logical location such as "network" may be specified. This scheme requires the existence of a run-time binder to manage the export and import of interfaces, this binder is called the DSL Trader†. A server exports its interface to the DSL Trader and a client imports an interface from the Trader, in this way the Trader is solely responsible for matching imports with exports. The Trader is at liberty to use any algorithm it choses to match import and export requests, and it is the properties of the algorithm chosen which allow for the effective management of heterogeneity in this architecture. The two essential components of the algorithm used are described below:

- A given interface may have multiple implementations, the Trader choses the implementation exported by the location specified in the import request.
- If an exact interface match cannot be found, the Trader searchs its export database for different, but functionally equivalent, interfaces exported by the location specified in the import request.

The first component allows different hardware and software to provide the same functionality without the importer being aware of these differences. The second component uses a set of rules to identify functionally equivalent interfaces and to match an import to a functionally equivalent export if no exact match exists. The rules for functional equivalence are described in detail in the section on "Functional

† The term trader is taken from the ANSA[ANSA1989] project's terminology, and refers to an extended name server which manages typed interfaces rather than uninterpreted strings.

Equivalence". A simple example will illustrate the usefulness of this algorithm. Consider the situation where a given location exports a video phone interface, this location clearly has the capability for point-to-point voice and video connection. If an import request is made on this location for a simple point-to-point phone conversation then the import should succeed since the location in question supports a superset of the required functionality.

Finally, DSL provides a mechanism for aggregating interfaces to build compound devices. This facility is providing to allow the re-use of existing interfaces and implementations in order to provide a much shorter development time for new and experimental applications.

The following sections examine the interface structure, functional equivalence and aggregation mechanisms in greater detail.

5.1 DSL Streams: Plugs and Sockets

A DSL stream as specified in an interface is a stream end point, i.e. it can be the source or sink of a stream. For this reason the DSL stream component consists of plugs (stream sources) and sockets (stream sinks). A plug must be connected to a socket in order to create a stream over which data can flow. A plug or socket is named and a name can be used once within the same interface. Each plug or socket is typed by a stream type. A stream type consists of a stream type name and the following properties:

- QOS properties supported, including the format of the stream synchronisation information.
- The synchronisation points supported, this will be a list of upcalls and their associated arguments which can be registered with this stream. Each such point will have a synchronisation variable associated with it.

Stream type checking is based solely on the name given to the stream type. Therefore for a stream to be successfully created the plug and socket must be of the same stream type, that is the stream types they specify must have the same name. The following diagram shows a stream of type "VideoStream", created from a plug and socket of the same stream type; the plug is called "Camera" and the socket "Display". The stream, plug and socket types are given within their representative shapes, with their names appearing below the shapes.

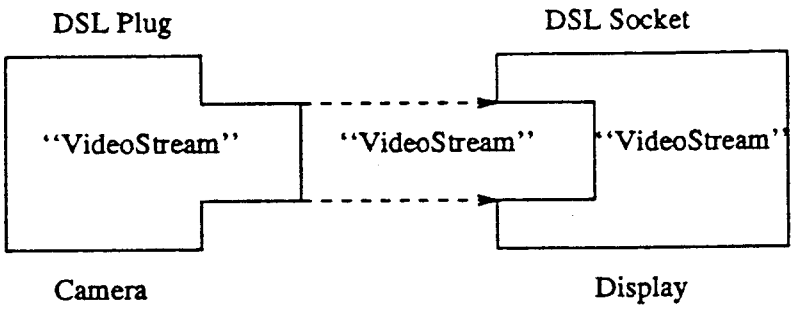

Figure 3. Video Stream

If the stream types of the plug and socket do not match it may be possible to use a translator interface, this is simply an interface with a socket of the same stream type as the original plug and a plug of the same stream type as the original socket. Translators are found by interrogating the DSL Trader. Figure 4 shows an audio stream created using a translator. In this case the end point plug is of type "Audio A-Law" whilst the end point socket is of type "Audio Mu-Law"; the translator has a socket of type "Audio A-Law" which is directly connected to a plug of type "Audio Mu-Law".

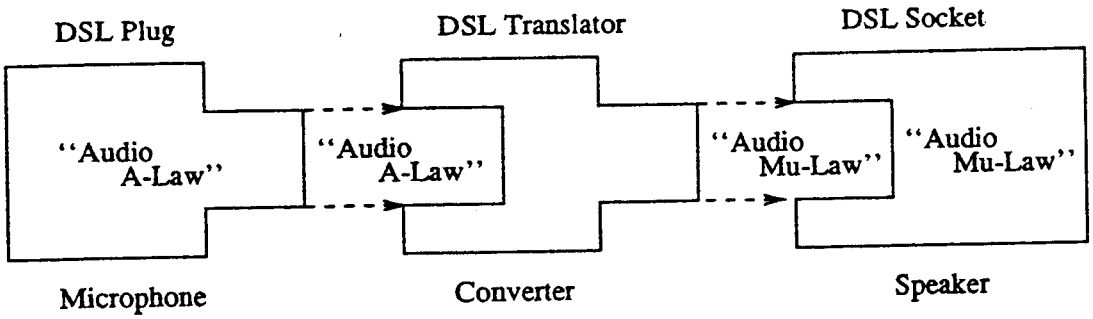

Figure 4. Translated Audio Stream

5.2 DSL Operations

The operations component contains all the operations available for controlling the streams supported by this interface as well as a small set of management operations supported by all interfaces. Each operation is named, and a name can only occur once in the same interface. The management operations are present to provide a uniform means of managing stream connections across all devices, this includes the establishment of connections and access and manipulation of the QOS properties. The remaining operations are entirely

interface specific, an operation takes a set of arguments and returns a set of results.

Sample interfaces for simple camera and display devices are given below. The syntax used is only intended to give a general view of the structure of an interface; it is in no way fully defined or finalised. For simplicity the following examples do not demonstrate the synchronisation of related streams.

```
Video: STREAMTYPE WITH QOS [ xres, yres: INTEGER ],
    SYNCHRONISATION POINTS { Suspended, Resumed, Stopped, Error };

INTERFACE CameraDev =

    Camera: STREAMPLUG OF STREAMTYPE Video;

    MANAGEMENT OPERATION ConnectTo [ address: ProtocolAddress,
        xres, yres: INTEGER ] RETURNS [ BOOLEAN ];

    OPERATION Start [] RETURNS [];
    OPERATION Suspend [] RETURNS []; -- temporarily pause video
    OPERATION Resume [] RETURNS []; -- resume video
    OPERATION Stop [ blocking: BOOLEAN ] RETURNS [];

END

INTERFACE DisplayDev =

    Display: STREAMSOCKET OF SREAMTYPE Video;

    MANAGEMENT OPERATION ListenTo [ address: ProtocolAddress,
        xres, yres: INTEGER ] RETURNS [ BOOLEAN ];

    OPERATION DisplayOn [ xpos, ypos: INTEGER ] RETURNS [];
    OPERATION DisplayOff [] RETURNS [];

END
```

Figure 5. Simple Camera and Display Interfaces

The synchronisation points defined for the video stream will be generated in response to the corresponding `CameraDev` operations, or error; the `Stop` operation takes a flag stating whether blocking or non-blocking synchronisation is required. The physical camera device may have pause and stop controls which can also generate these synchronisation points.

The set of management operations shown in the examples is not complete; work is underway to determine a full set of such operations and to automate their use to the greatest degree possible. This automation may take the form of a set of library procedures. The provider of an interface implementation must implement the management operations along with all the other operations defined in the interface, again a library of common implementations will ease this task.

The following pseudocode illustrates how the above interfaces could be used to realise a unidirectional video stream between two specified locations.

```
-- First import the two interfaces

    cameraHandle := IMPORT CameraDev AT locationX
    displayHandle := IMPORT DisplayDev AT locationY

-- Connect is a library procedure which in turn invokes the
-- management operations provided by the devices

    Connect( cameraHandle, displayHandle, xres, yres )

-- Now the devices are ready for use

    cameraHandle.Start()
    displayHandle.On( 0, 0 )

-- Fork a thread to wait for user input, when input is received
-- call cameraHandle.Stop(), Suspend() or Resume() as required

    Loop

-- wait for synchronisation points to be reached,

        WaitForSV( Suspended SVOR Stopped, ForEver )

        if( TestSVC( Stopped ) )
            BREAK;
        else
            WaitForSv( Resumed, ForEver )

    EndLoop

    displayHandle.Off()
    cameraHandle.Stop( TRUE )
```

Figure 6. Use of Camera and Display Devices

The above example gives a general feel of the style of programming required to drive DSL devices, more practical work is required to fully implement this programming model. The mechanism used to listen for user input depends on the input/output system being used; the essential point is that the uniform synchronisation mechanism allows the programmer to treat the video stream as structured, without regard to how this structure is imposed. Note that the user who has access to the camera is not necessarily the same user who has access to the controlling application's input. In this example the user receiving the

video stream (i.e. the display end) can have access to this input and thus suspend, resume or stop the video stream, as a result both users have access to the same control interface.

5.3 Functional Equivalence

Functional equivalence is essentially the same as the notion of conformance as used in the Emerald system,[Black1986] with a simple extension to deal with the streams component of a DSL interface. Conformance is preferred to inheritance as used in the Smalltalk system since it expresses a relationship between interfaces, whilst inheritance is a relationship between implementations. Informally the rules for functional equivalence in DSL are as follows:

An interface S is functionally equivalent to an interface T (written $S \leq T$) if and only if the following conditions hold.

i. S provides at least the plugs and sockets of T (S may have more).

ii. For each plug or socket in T, the corresponding plug or socket in S is of the same stream type.

iii. S provides at least the operations of T (S may have more).

iv. For each operation in T, the corresponding operation in S has the same number of arguments and the same number of results.

v. The types of the arguments of T's operations conform to the types of the arguments of the corresponding operation in S (i.e. the arguments must conform in the opposite direction to the interfaces)†

If these conditions are met then an import request for interface T can be satisfied with an export of interface S.

---

† Conformance as applied to the data types of the arguments is identical to that used in Emerald.

5.4 Aggregation

The aggregation facility allows compound interfaces to be constructed from existing interfaces, a compound interface consists of a set of sub-interfaces plus a streams and operations component. The functional equivalence rules given above apply to simple interfaces, i.e. interfaces which do not contain any sub-interfaces. The rules for functional equivalence can be extended to cope with compound interfaces as follows.

A compound interface S is functionally equivalent to an compound interface T if:

i. The streams and operations component of S are functionally equivalent to the streams and operations component of T, as given above.

ii. For each sub-interface in T there is a corresponding sub-interface in S which is functionally equivalent to the sub-interface in T.

iii. Apply rules i) and ii) recursively for all the sub-interfaces in T and corresponding sub-interface is S.

6. The Architecture

This section places the previous discussions on synchronisation and managing heterogeneity into a uniform architectural model, thus providing a consistent and precise design framework within which the system designer and application writer can work.

A layered approach is taken to decomposing this architecture into its functional components, three such layers exist as shown in figure 7 .

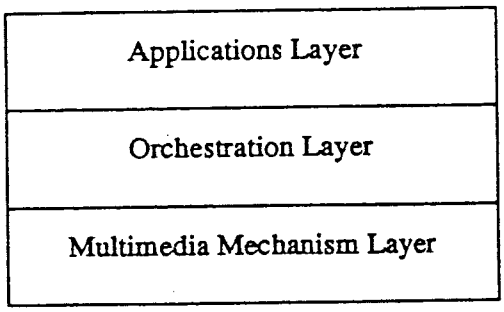

Figure 7. Architectural Layers

The multimedia mechanism (MMM) layer includes the generation, transport and presentation of real-time multimedia streams. The transport function has previously been referred to as the communications subsystem. The interface to this transport component is specified by the management operations in a DSL interface. The generation component deals with the generation of stream samples, i.e. physical synchronisation frames, whilst the presentation component accepts PSFs and presents them to the user. The generation and presentation components present two interfaces, one to the transport function and one to the Orchestration layer. The interface to the transport function is in terms of PSFs and is provided for efficiency reasons, in particular to minimise jitter. The use of this "sideways" interface must be under the control of the Orchestration layer, thus preserving the layering of control functions whilst allowing for the sideways movement of data. The interface to the Orchestration layer is specified in terms of LSFs. All of the MMM interfaces are specified using DSL, with some components of the interface being implemented by the transport function and other components by the generation and presentation components. Figure 8 shows the MMM structure in more detail. Note that the horizontal arrows indicate data flow, whilst the vertical arrows indicate data and control flow.

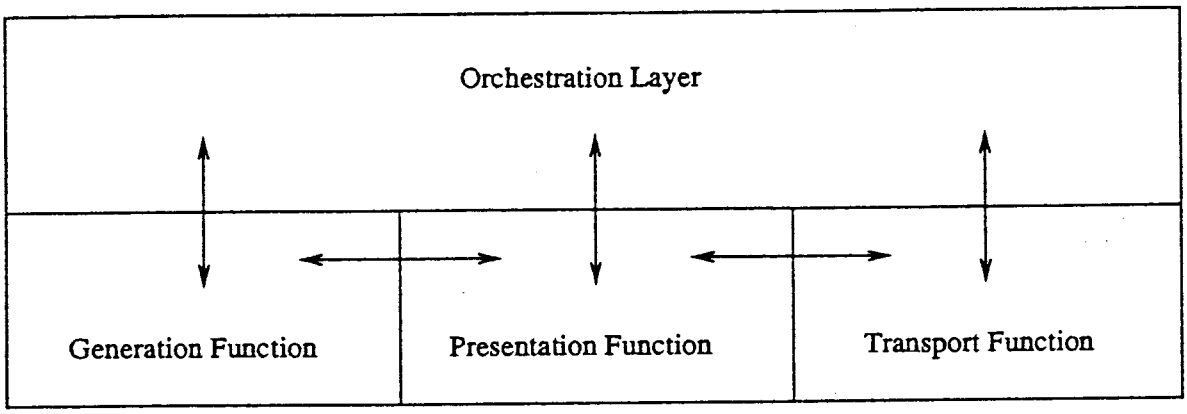

Figure 8. Detailed Multimedia Mechanism Structure

The Orchestration layer performs the two distinct functions of interfacing the application to the MMM and of implementing the heterogeneity management mechanism. The first function requires the implementation of the synchronisation scheme presented above whilst the second requires the implementation of DSL and the DSL Trader. DSL is the glue that binds the functional components of the architecture together, that is, it is used to specify all the interfaces present in the system.

Finally, the application layer, previously referred to as the Distributed Computing System, contains the controlling application. It can now be seen that the Orchestration layer provides the integration between the DCS and communications subsystem. This is illustrated in figure 9 .

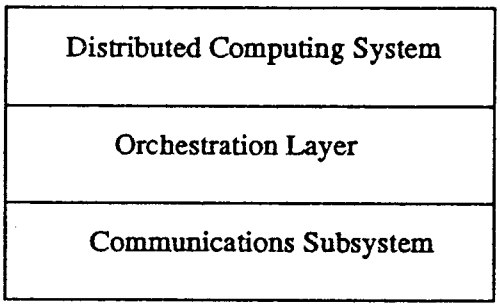

Figure 9. Orchestration Layer Integration

Figure 10 shows the relationship between this multimedia architecture and the OSI Reference Model. There are two main differences between the two, firstly the multimedia architecture layers control interfaces whilst the OSI Reference Model layers control and data interfaces. Also, the two take a radically different approach to managing heterogeneity, the OSI Reference Model assumes that interworking is managed at

the lower levels of the model and that the higher levels need not be aware of any lower level differences, whereas the multimedia architecture provides explicit architectural support for managing heterogeneity and this management is based on out-of-band control techniques.

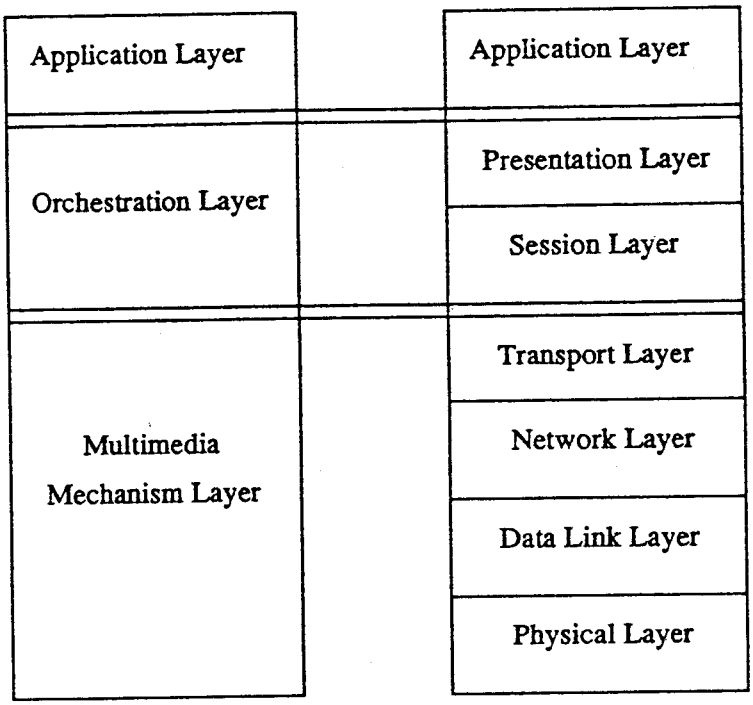

Figure 10. Multimedia Architecture and OSI Reference Model Relationship

7. Work Plan

Work is currently in progress to implement this multimedia architecture. This work includes extending a prototype multi-service protocol suite (the MSNL protocol suite)[McAuley1989] to implement the QOS, synchronisation data gathering and notification functions described above. This will then be integrated into a prototype DCS, namely the ANSA[ANSA1989] testbench; this system provides light-weight threads, RPC and distributed naming over a variety of operating systems. Both MSNL and the ANSA testbench run over the UNIX™ operating system. MSNL runs over both Ethernet and the Cambridge Fast Ring local area networks. Some applications will be built to test the utility of this architecture, these could include a simple video phone and associated call management, simple voice and video recording and playback.

8. Summary

The issues involved in providing real-time multimedia communication have been discussed in detail. Extensions to existing communications subsystems and distributed computing systems have been suggested which will enable them to better meet the requirements of multimedia communication. In particular solutions to the problems of synchronisation and heterogeneity have been described in detail. These solutions have been incorporated into an architecture which provides a set of design rules and guidelines within which both the system implementor and application writer can work. The interfaces specified by the architecture are all described using an interface specification language (DSL).

References

ANSA1989. ANSA,, *ANSA Reference Manual*, Release 01.00 March 1989.

Ades1986. Ades, Stephen, Roy Want, and Roger Calnan, "Protocols for Real Time Voice Communication on a Packet Local Network," pp. 525-530 in *Proceedings of the International Conference on Communications*, IEEE, Toronto (June 1986).

Anderson1988. Anderson, David P., "A Software Architecture for Network Communication," pp. 376-383 in *Proceedings of the Eigth International Conference on Distributed Computing Systems*, IEEE (June 1988).

Black1986. Black, Andrew, Norman Hutchinson, Eric Jul, and Henry Levy, "Object Structure in the Emerald System," pp. 78-86 in *OOPSLA '86 Proceedings*, ACM (September 1986).

Buxton1985. Buxton, William, Ralph Hill, and Peter Rowley, "Issues and Techniques in Touch-Sensitive Tablet Input," pp. 215-224 in *SIGGRAPH '85*, ACM, New York (July 1985).

Buxton1986. Buxton, William and Brad A. Myers, "A Study in Two-handed Input," pp. 321-326 in *Proceedings HCI'86 Conference on Human Factors in Computing Systems*, ACM, New York (April 1986).

Calnan1987. Calnan, Roger S., "ISLAND: A Distributed Multimedia System," paper (November 1987).

Christodoulakis1986. Christodoulakis, S., M. Theodoridou, F. Ho, M. Papa, and A. Pathria, "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System," *ACM Transactions on Office Information Systems* 4(4) pp. 345-383 (October 1986).

Herman1987. Herman, Gary, Michael Ordun, Christine Riley, and Leland Woodbury, *The Modular Integrated Communications Environment (MICE): A System for Prototyping and Evaluating Communications Services*, Bell Communications Research, Morristown, New Jersey 07960, USA (March 1987).

Hill1986. Hill, Ralph D., "Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction-The Sassafras UIMS," *ACM Transactions on Graphics* **5**(3) pp. 179-210 (July 1986).

Hudson1987. Hudson, Scott E., "UIMS Support for Direct Manipulation," *Computer Graphics* **21**(2) pp. 120-124 (April 1987).

Lantz1987. Lantz, Keith A., Peter P. Tanner, Carl Binding, Kuan-Tsae Huang, and Andrew Dwelly, "Reference Models, Window Systems and Concurrency," *Computer Graphics* **21**(2) pp. 87-97 (April 1987).

Lazar1985. Lazar, Aurel A., Avshalom Patir, Tatsuro Takahashi, and Magda El Zarki, "MAGNET: Columbia's Integrated Network Testbed," *IEEE JSAC* **SAC-3**(6) pp. 859-871 (November 1985).

Lazar1986. Lazar, Aurel A., Mark A. Mays, and Kenichi Hori, "A Reference Model for Integrated Local Area Networks," pp. 531-536 in *Proceedings International Conference on Communications*, , Toronto (June 1986).

Lazar1987. Lazar, Aurel A. and John S. White, "Packetized Video on MAGNET," paper, Center for Telecommunications Research, Columbia University, NY 10027, USA (July 1987).

McAuley1989. McAuley, Derek, "Protocol Design for High Speed Networks," PhD thesis, University of Cambridge Computer Laboratory (September 1989).

Naffah1986. Naffah, Najah and Ahmed Karmouch, "Agora-An Experiment in Multimedia Message Systems," *IEEE Computer* **19**(5) pp. 56-66 (May 1986).

Newman1989. Newman, Peter, "Fast Packet Switching for Integrated Services," PhD Thesis, University of Cambridge Computer Laboratory (March 1989). Technical Report No. 165

Newman1988. Newman, R. M., Z. L. Budrikis, and J. L. Hullet, "The QPSX Man," *IEEE Communications Magazine* **4**(26) pp. 20-28 (April 1988).

Nicholson1985. Nicholson, Robert T., "Usage Patterns in an Integrated Voice and Data Communications System," *ACM Transactions on Office Information Systems* 3(3) pp. 307-314 (July 1985).

Poggio1985. Poggio, A., J. J. Garci Luna Acheves, E. J. Craighill, D. Moran, L. Aguilar, D. Worthington, and J. Hight, "CCWS: A Computer-Based, Multimedia Information System," *IEEE Computer* 18(10) pp. 92-103 (October 1985).

Postel1988. Postel, Jonathan B., Greogory G. Finn, Alan R. Katz, and Joyce K. Reynolds, "An Experimental Multimedia Mail System," *ACM Transactions on Office Information Systems* 6(1) pp. 63-81 (January 1988).

Redman1987. Redman, Brian E., "A User Programmable Telephone Switch," paper, Bell Communications Research, Morristown, New Jersey 07960, USA (April 1987).

Root1986. Root, Robert W. and Steve D. Hawley, *DynaMICE: A Direct Manipulation Graphics Interface for Controlling Advanced Telecommunications Services*, Bell Communications Research, Morristown, New Jersey 07960, USA (1986).

Ross1986. Ross, Floyd E., "FDDI - A Tutorial," *IEEE Communications Magazine* 24(5) pp. 10-17 (May 1986).

Schneiderman1983. Schneiderman, Ben, "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer* 16(8) pp. 57-69 (August 1983).

Swinehart1983. Swinehart, D. C., L. C. Stewart, and S. M. Ornstein, "Adding Voice to an Office Computer Network," in *Proceedings IEEE Globecom 1983*, IEEE (November 1983). also available as Xerox Palo Alto Reseach Center Technical Report CSL-86-1, June 1986

Swinehart1987. Swinehart, Daniel C., Douglas B. Terry, and Polle T. Zellweger, "An Experimental Environment for Voice System Development," *IEEE Knowledge Engineering* 1(1) pp. 39-48 (February 1987).

Tanner1987. Tanner, Peter P., "Mutli-Thread Input," *Computer Graphics* 21(2) pp. 142-144 (April 1987).

Temple1984. Temple, Steven, "The Design of a Ring Communication Network," PhD thesis, University of Cambridge Computer Laboratory (January 1984). Technical Report No. 52

Thomas1985. Thomas, Robert H. and Harry C. Forsdick et al, "Diamond A Multimedia Message System Built Upon a Distributed Architecture," *IEEE Computer* 18(11) pp. 1-31 (November 1985).

Want1988. Want, Roy, "Reliable Management of Voice in a Distributed System," PhD thesis, University of Cambridge Computer Laboratory (July 1988). Technical Report No. 114

Tennenhouse1989. Tennenhouse, David L., "Layered Multiplexing Considered Harmful," in *Protocols for High Speed Networks*, , IBM Zurich Research Lab. (May 1989). IFIP WG6.1/6.4 Workshop