An Architecture for Real-Time Multimedia Communication Systems

COSMOS NICOLAOU

Abstract—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 synchronization of related real-time data streams, and the management of heterogeneous multimedia hardware. The synchronization problem is tackled by defining explicit synchronization properties at the presentation level and by providing control and synchronization 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 communications 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 prototype implementation conforming to this architecture is currently under construction.

I. 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 [2], [13]. Some work has also been done on extending the OSI reference model to cope with multimedia communication[14].
- Work on multiservice 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 [31], [8], [22], [17], [20], [21].
- Control of PABX functions from a computer system. These systems allow application programs to be written which control and customize the behavior of the PABX in question [24], [9], [23].
- The integration of voice communication into a digital network and distributed computing system. These systems allow for the implementation of software PABX’s, voice editing and storage, and multimedia (text and voice) document preparation, as well as the real-time transport of voice over a digital network [32], [7], [27], [28].
- 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 and video, and on the design and implementation of a special purpose workstation to handle the presentation of these media [15].

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 PABX’s. However, as network [30], [25], [18], [19] and CPU capacity increase, it is becoming possible to handle video as effectively as voice. These higher capacity networks and CPU’s 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 summarize, 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 synchronizing these multiple streams in order to bring about some meaningful communication, while the drive towards open systems carries with it the requirement to effectively manage heterogeneity. The architecture presented here directly addresses the issues of synchronization and heterogeneity.

The rest of this paper is structured as follows. A survey of the requirements of real-time multimedia communica-

0733-8716/90/0400-0391$01.00 © 1990 IEEE
tion and of the distributed computing system is presented along with a discussion of the interrelationships of these two sets of requirements. The issues of synchronization and heterogeneity are then discussed in detail, followed by a description of the architecture itself and its relationship to the OSI reference model.

II. 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 realize 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 packetization delay, the network transmission delay and the presentation delay. The packetization 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 packetization 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 packetization delay and also to give acceptable network utilization. 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 synchronization 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 acknowledgment packets to detect lost packets at the source, since a retransmitted packet following a timeout on an acknowledgment 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 4 ms 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 [2] 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 [15].

For any stream, with a given 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 a 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 prioritizing packets for the real-time stream in order to minimize queueing delays and therefore jitter, and also using a light weight protocol which does not use acknowledgment 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 [33] 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 [3] takes essentially this QOS approach to precisely tailor the behavior of the communications subsystem to the requirements of the user application.

The DCS as 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.

A. 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 synchronization. An
Both individual and separate but related streams at the QOS parameter, whose properties indicate the nature and mechanisms, whilst at the same time supplying the application is then able to implement fine level at the time it is required) to implement synchronization of both individual and separate but related streams at the (probably coarser) granularity with which the controlling application is best able to manage.

III. 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 synchronization 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 synchronization primitives must be efficiently implemented.

If the DCS is to be used to implement control and synchronization 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 synchronized, 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 synchronized to them, implying that the streams must be structured so as to allow for this synchronization.

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

IV. PRESENTATION LEVEL SYNCHRONIZATION

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

A. Lip-Synching

Lip-synching refers to the synchronization of spoken voice with the movement of the speaker’s lips. This synchronization 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 synchronization of these related streams. Given that bounded jitter is achievable it is possible to construct a buffering scheme which maintains the synchronization of the individual streams over relatively short (minutes) periods of time. It is then left to the application to ensure that these streams remain synchronized with respect to each other over longer periods of time. Two pieces of information are required to implement this synchronization.

- 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 synchronization 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.

B. 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 UIMS’s which support this concurrency [12]. This drive is motivated by the
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 its 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 synchronization points, since every time such an error occurs synchronization is lost and some action must be taken to resynchronize or to abandon communication in some graceful manner. It is also useful to consider all exceptional, though not necessarily erroneous, events as synchronization points. For instance opening or closing a real-time connection may generate synchronization events when the first and last (respectively) samples are received. This allows for related streams to be synchronized with respect to each other whenever such a synchronization point is reached. In the lip-synching example the controlling application may wish to wait until both streams reach their last sample synchronization 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 restart the stream. These synchronization 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.

C. A Synchronization Scheme

This section presents a scheme for implementing the three types of essential synchronization which have been identified by the previous examples. The first, referred to as isochronous synchronization is concerned with maintaining the real-time synchronization of related streams. The second and third deal with synchronization 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 synchronization is itself an exceptional event requiring some resynchronization 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 synchronization frames (PSF). At the next level the stream is structured as an ordered sequence of logical synchronization frames (LSF), each of which consists of a number of physical synchronization frames. PSF's are intended as the unit of synchronization within the communications subsystem, whereas LSF's are the unit of synchronization 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 synchronization 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 synchronization it requires in a convenient manner.

Fig. 1 illustrates how this synchronization 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 synchronization points at both levels in the stream, though only the synchronization points occurring at LSF boundaries are communicated to the controlling application.

Given this synchronization scheme, it is then necessary...
to define how the synchronization points are indicated to
the controlling application, how operations on these
streams are synchronized with respect to these streams,
and how synchronization information gathered by the
communications subsystem is presented to the controlling
application. 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, while the converse is not true.

Each stream will have an associated set of stream spe-
cific operations. These operations must be synchronized
with respect to the streams to which they apply, this syn-
chronization is defined in terms of the streams LSF’s. 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 does not take effect within the stated num-
ber of LSF’s then the communications subsystem must
report a timeout error. The benefits of specifying this syn-
chronization relationship is that both the application writer
and the stream implementor have a precise synchroniza-
tion model within which to work, thus eliminating poten-
tial confusion.

The following figure (Fig. 2) shows how a ‘stop’ op-
eration would be synchronized 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 op-
eration in question is blocked until completion, alterna-
tively the operation may return immediately (i.e., is not
blocked) and an explicit synchronization event will be
generated on its completion. Both blocking and nonblock-
ing modes are illustrated below.

The synchronization information gathered by the com-
munication 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 synchronization event.

The first situation can be dealt with by defining these
regular intervals as synchronization 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.

D. Synchronization Summary and Complete Example

We now have a two-level synchronization scheme with
upcalls occurring at synchronization points in the upper
of the two levels. A procedural interface is provided for
setting and modifying QOS properties and for obtaining
synchronization 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 as 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 asso-
ciated video. The QOS properties can be used to define
an isochronous upcall at some rate, this upcall can mon-
tor the synchronization of the stream being played and
update the time line. If the stream is found to be losing
synchronization then this upcall can take corrective ac-
tion; 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 synchronization points are defined in
LSF’s, 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 in-
form the user of the error. The following section suggests
how this scheme can be integrated into the DCS.

E. Integrating Stream Synchronization into the DCS

Concurrent activities typically require synchronization
points, each such point is represented as a synchroniza-
tion variable. Such a variable, if set, indicates that the
synchronization point has been reached, if unset then this
point has not been reached. Synchronization variables
(SV) are stream specific and are represented as a triple
(stream, synchronization point, value). Three primitives
are defined which operate on these variables.
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 chooses 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 searches 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 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 provided to allow the reuse 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.

A. 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 synchronization information.
- The synchronization 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 synchronization 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. Fig. 3 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.

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. Fig. 4 shows an audio stream created using a translator. In this case the end point plug is of type "Audio A-Law," while 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."

B. 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 in Fig. 5. 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 finalized. For simplicity the following examples do not demonstrate the synchronization of related streams.

The synchronization 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 nonblocking synchronization is required. The physical camera device may have pause and stop controls which can also generate these synchronization 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 (Fig. 6) illustrates how the above interfaces could be used to realize a unidirectional video stream between two specified locations.
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 synchronization 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.

C. Functional Equivalence

Functional equivalence is essentially the same as the notion of conformance as used in the Emerald system, [4] 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, while 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 ≤ T) if an only if the following conditions hold.

1) S provides at least the plugs and sockets of T (S may have more).
2) For each plug or socket in T, the corresponding plug or socket in S is of the same stream type.
3) S provides at least the operations of T (S may have more).

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

5) 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).5

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

D. Aggregation

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

A compound interface S is functionally equivalent to a compound interface T if the following holds.

1) The streams and operations component of S are functionally equivalent to the streams and operations component of T, as given above.
2) For each subinterface in T there is a corresponding subinterface in S which is functionally equivalent to the subinterface in T.
3) Apply rules 1) and 2) recursively for all the subinterfaces in T and corresponding subinterface is S.

VI. THE ARCHITECTURE

This section places the previous discussions on synchronization 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 Fig. 7.

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 synchronization frames, whilst the presentation component accepts PSF’s 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 PSF’s and is provided for efficiency reasons, in particular to minimize jitter. The use of this “sideways” interface must be under the control of the Orchestration layer, thus preserving the layering of con-

5Conformance as applied to the data types of the arguments is identical to that used in Emerald.
control functions while allowing for the sideways movement of data. The interface to the Orchestration layer is specified in terms of LSL’s. 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. Fig. 8 shows the MMM structure in more detail. Note that the horizontal arrows indicate data flow, while the vertical arrows indicate data and control flow.

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 synchronization scheme presented above while 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 Fig. 9.

Fig. 10 shows the relationship between this multimedia architecture and the OSI reference model. There are two main differences between the two, first, 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.

VII. WORK PLAN

Work is currently in progress to implement this multimedia architecture. This work includes extending a prototype multiservice protocol suite (the MSNL protocol suite) [16] to implement the QoS, synchronization data gathering and notification functions described above. This will then be integrated into a prototype DCS, namely the ANSA [1] 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.

VIII. 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 synchronization 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
