AN ARCHITECTURE SUPPORTING MULTI-MEDIA INTEGRATION

Joseph S. Sventek

ANSA Project, 24 Hills Road, Cambridge CB2 1JP, United Kingdom

Abstract

An architecture supporting the construction of integrated, multi-media applications is presented. The main characteristics of such applications are described first. A model which permits the I/O devices associated with a multi-media station to be incorporated into an object model for distributed systems is then presented. The architectural features required to permit temporal synchronization of the data from the different devices are then discussed. The utility of the architecture is established through its application to several examples. Future areas of investigation are outlined.

1 Introduction

Historically, humans have used different media as vehicles for interpersonal communication. The human voice has proved most appropriate for communication in which there is a high degree of interaction between the communicants. Visual cues are often used to embellish the meaning of associated audio information; in addition, visual information can provide the primary means of communication, e.g., mime. The written word permits the transfer of large volumes of information, without the need for interaction between the writer and the reader.

Each of these forms of communication has been technologically enhanced during the last century. The oldest innovation, the telephone, permitted the extension of audio communication to individuals in different places; this enhancement was accompanied by the loss of the visual stimuli mentioned above. The invention of television provided the capability for visual communication between people in different places. Finally, the advent of computing systems has provided an electronic means for communicating the written word; the functionality of such systems spans the spectrum from providing access to large, static databases (e.g., electronic encyclopaedias) to supporting interactive textual exchange (e.g., electronic mail).

The decreasing cost and increasing power of computer systems have led to a desire to integrate these various communication capabilities. Integrated use of these media promises to open up a whole new range of communication methods. The independent development of the different communication technologies has led to significant technical difficulties which must be overcome before such integrated communication capabilities can be realized.

The mechanization of voice and video communications has been achieved, in each case, by the transmission of a clocked, continuous stream of data. The communication medium over which the data is transmitted must have sufficient bandwidth to permit the sender and receiver of the data to remain synchronized. This is usually provided by employing circuit-switching techniques - i.e., a physical path of sufficient bandwidth is dedicated to the communicating pair for the duration of the interaction. It is also the case that there is sufficient redundancy in voice and video signals to permit the data to be sent unreliably, although modern circuits are actually quite reliable.

For the transmission of data between computer systems, on the other hand, the emphasis has been placed upon obtaining reliability at the expense of guaranteed transmission bandwidth. Such systems are often based upon packet switching (many conversations share the same physical path), and normally employ a sophisticated software structure to provide reliable transmission.

Any attempt to integrate these media will necessarily be concerned with coordinating data with very different delivery requirements. Each element of a multi-media application system must be able to schedule itself to meet the timing
requirements of any clocked voice/video channels; it must also be able to synchronize the transmission of data from the different media to permit their replay to reflect the temporal relationships at input, e.g. if voice is used to annotate a visual action, the receiver should certainly not hear the annotation before seeing the visual operation. This paper explores an architectural foundation upon which such integrated, multi-media applications can be built.

The efforts herein described form a small part of the distributed system architecture under development by the Advanced Networked Systems Architecture (ANSA) Project. This project constitutes a collaborative effort to develop an open architecture standard that will support truly integrated, multi-vendor distributed systems. The overall project is described in reference 1.

1.1 Levels of integration

It is important to be clear about what is meant by multi-media integration. In an abstract sense, this term has been used to describe integration at three logical levels:

- the physical level - the data from the separate media are multiplexed over a single physical connection
- the service level - the interactions between an element of a multi-media application and its media are achieved through a common, service interface; the interactions between the elements of the application are also via the common service interface
- the human-interface level - this describes the presentation of the different media to the user

Since the construction of multi-media applications is of primary interest to ANSA, this paper is concerned with integration at the service level. The impact of integration at the physical level upon the design of multi-media applications is discussed in §7.

1.2 Requirements of multi-media applications

Figure 1 is a conceptual picture of a typical multi-media application. Two (or more) application entities, each of which is able to send/receive data from a set of media devices (keyboard/display, telephone, video camera, etc.), share their respective data in an application-specific manner.

Any architecture for building such application systems must satisfy three requirements:

- it must provide a way for an application entity to interact with its devices
- it must permit an application entity to preserve the temporal relationships between the data from the separate media
- it must provide a way for the application entities to share the correlated information

An architecture satisfying these requirements must do so in a general way, since the specific needs of individual applications with respect to each of these requirements may be different. For example, a particular application may be able to dispense with the need for temporarily correlating the information.

2 An object model for distributed systems

A distributed system consists of components which are logically separate, i.e. their separation is defined without reference to placement or physical realization. Each interaction between logically separate components is termed an operation, defined as an elementary cycle of information flow which comprises at least a delimited information flow from a source to a destination and at the most that flow coupled with a delimited information flow back in reply.
In an object model of a distributed system, the only style of interaction permitted between components (objects) is the invocation of operations. For a given object, there will be some set of operations which models all possible visible behaviours. Objects interact through an interface, the specification of which defines the set of permissible operations. An interface specification consists of three parts:

- **the operation type** - the invariant syntax and semantics of an operation
- **the interaction type** - the functional attributes of the dynamic behaviour of an operation
- **the interaction mode** - the quality attributes of the dynamic behaviour of an operation

The interaction type is used to describe the fundamental communication requirements of the interface, e.g. the use of a clocked channel for voice, whether an operation will result in a reply, whether the interface is master/slave or peer-to-peer. The interaction mode indicates the desired quality of communication, permitting a choice to be made among several options which satisfy the fundamental requirements dictated by the interaction type. Examples of such quality attributes include performance, reliability, security and cost.

In order for logically separate objects to interact, they must bind themselves to the abstract interfaces through which they wish to communicate. An object wishing to make its interface available to other objects must **export** the interface definition; an object wishing to use an interface must **import** the interface definition. The resulting indirect binding permits the participating objects to interact using the operations defined in the interface specification.

The operations which can be invoked on an indirect binding fall into three general categories:

- **reaction** - the response, of whatever nature, to an externally applied stimulus (event). For conventional program-program interactions, this might be more appropriately named **execute**
- **interrogation** - an operation which stimulates the immediate invocation of a reaction operation in the cooperating object. The invoking object waits for the reaction to complete, and the results of the reaction are returned to the invoking object
- **announcement** - an operation which may eventually stimulate the invocation of a reaction operation in the cooperating object. The invoking object does not wait for the reaction to complete

The time sequence diagram in figure 2 illustrates an interaction in which both interrogations and announcements are used. Time is increasing as one scans down the figure. Note that there is no explicit termination of the dialogue; this is a characteristic of indirect bindings.

The requirements of specific applications may dictate concurrent interactions of an object over several bindings. The blocking nature of interrogations dictate that each binding be manipulated by a separate thread. A thread is an abstraction intended to capture the notion of an active agent in a computation, e.g. threads may correspond to processes, tasks, co-routines or event queues in actual implementations. A program begins execution as a single thread; additional threads may be created and destroyed explicitly by
means of fork, join and halt primitives, or implicitly by means of a cobegin ... coend structure.

Building a distributed system based upon this architecture thus entails a sequence of steps:

1) define the objects which make up the system
2) define the interfaces which those objects make available to other objects
3) execute the exports and imports of these interfaces to achieve the required indirect bindings, with the correct interaction types and interaction modes
4) make the appropriate sequences of operation invocations over the indirect bindings to achieve the desired, application-specific ends.

3 An object model for device interaction

The first requirement on a multi-media architecture is to provide a stylized way for the application to interact with its devices. Given the overall system model presented in §2, we need to provide a device model in which the interactions with physical devices can be described in terms of objects, interfaces and bindings.

3.1 Simple device model

In its most general form, a device presents two bi-directional channels to the outside world, a control channel for manipulating the state of the device and a data channel for sending/receiving data between the device and the controlling system. The data sent/received are interpreted subject to the current state established through the control channel. We will call this pair of bi-directional channels a pipe. Obviously, for many devices, one or more of the four possible directions does not exist, e.g. the output data channel to a mouse makes little sense.

In order to provide an object/interface view of device interaction, each device is controlled by a transformer object. A transformer packetizes the data from the control and data channels of a device, placing the resulting events in a single logical queue. The operations provided by the transformer interface are:

- ReceiveNextItemFromQueue (i/f, item, type)
- SendControlMessage (i/f, msg)
- SendDataMessage (i/f, msg)

where the formal parameters have the following meanings: i/f is the interface which was imported, item is replaced by the next item from the queue, type is replaced by one of the set (Control, Data) and msg is the message to send.

The serialization of the control and data traffic into a single logical queue is important, since it preserves the temporal relationships between the activities on the control and data channels.

This model enables an application object to interact with its devices. Figure 3 shows an example for one such application with N devices. Each thread invokes operations in the transformer object to which it is bound, with the underlying system scheduling the threads independently.

![Application Object](image)

**Figure 3.** An application with N devices

3.2 Interconnecting devices

It is often desirable to splice the data channels of two devices together, i.e. the output of one becomes the input of another, and vice versa. While this can be achieved through program intervention (by continuously receiving the next item from one queue and sending it to the associated device), it should be possible to splice the data channels without incurring the overhead of program intervention. A connector is an object which can be inserted between two devices to achieve this splicing.

Since the only way a program can interact with a device is through the object interface provided by its transformer, the following two operations must be included in the transformer interface:

- InsertConnector (i/f, other-i/f, RetainDataChannel)
- RemoveConnector (i/f, other-i/f)

The parameter i/f is as before; the other-i/f
3.3 Examples

Connectors are useful only if the devices being connected interchange exactly the same types of information. An obvious application area is the establishment of a telephone call. A telephone management program in a workstation, for example, would import two device interfaces - one for the telephone itself, the other for the local exchange. Consider the case when a call is originated from the telephone. When the receiver goes off-hook, a control message to that effect will be delivered to the thread invoking a `ReceiveNextItemFromQueue` operation (through an interrogation) on the telephone's transformer interface. The program will immediately insert a connector between the telephone data channel and the exchange data channel (see figure 4), followed by an off-hook control message to the exchange interface. At this point, the exchange will generate dial tone and the user will interact with the exchange until the conversation has concluded. When the telephone is placed back in the cradle, an on-hook control message is sent to the telephone transformer interface. The program will send an on-hook control message to the exchange, followed by a removal of the connector between the data channels.

A different style of device interconnection, in which the control and data information can be modified to enhance their value, has been described. Such interconnection is best mirrored in our model through objects which export an interface identical to the transformer interface and pass the control and data messages to/from the actual transformer interface, adding/subtracting additional features and processing along the way.

4 Temporal synchronization of separate media

The device model described in §3 guarantees that the events from a particular device are serialized, preserving the temporal relationship between the activities on the control and data channels. Of equal importance to many multi-media applications is the synchronization of the events on two (or more) of the input media. As an example, consider an electronic blackboard application with voice annotation. As figures are drawn on the blackboard window, narrative describing the actions is provided vocally. The replay of the information must ideally exhibit the same temporal serialization. In particular, presentation of the voice annotation before the
blackboard manipulations which it describes is likely to lead to confusion on the part of the receiver.

There are two aspects of this temporal synchronization, thread scheduling constraints, which guarantee the temporal serialization of events from several devices, and data structuring, which can preserve these temporal relationships. Each aspect is treated separately below.

4.1 Scheduling of multiple threads

As described in §2, when a thread invokes an interrogation operation on a device interface to ReceiveNextItemFromQueue (RNIFQ), it will block until an item is available. Upon the occurrence of an event, the thread will be unblocked, and will be able to process the item. Until the processing is complete, we are guaranteed that no other event from that device will be processed.

Suppose that while one thread is processing an event, an event occurs which satisfies a blocked interrogation operation on a different thread. If the underlying system can pre-empt the currently executing thread in favour of the recently enabled one (because of a priority scheme, for example), then the temporal serialization of the two events can be lost (as exhibited by external behaviour from the application). To avoid this situation, we must require that the execution of a thread within an application be non-pre-emptable by another thread within the same application.

While non-pre-emption is a necessary condition, it is not sufficient. Consider the following pseudocode for the 'use if/else' box from figure 3.

```plaintext
LOOP
  ReceiveNextItemFromQueue
  "Process Item"
END
```

Assuming that 'Process Item' does not block, the only place where the thread can lose control of the processor is by executing an RNIFQ operation when there are no outstanding events on the device queue. This type of hogging is a well-known phenomenon in many areas of computer science; in this particular case, it prevents the application from being able to serialize the events from the different devices. It is clear that a stronger guarantee with regards to thread scheduling must be provided. A mechanism for providing this guarantee is shown in figure 5.

Each event from the logical queue associated with a device is merged into a single event queue for the entire object. Whenever the current thread releases the processor (by invoking an interrogation operation), the event at the head of the application queue is delivered (the thread blocked at an interrogation operation for that device is unblocked). This mechanism, coupled with the non-pre-emption requirement, guarantees that the temporal ordering of events from the different devices is maintained.

4.2 Data Structuring

In the previous section, little was said concerning the processing performed upon the return of an item from a device. In the best of situations, it might be expected that data items can be immediately displayed/transmitted. It will often be the case that the data or control information needs to be combined with information from another device to form a more coarse-grained data unit which has meaning to the application as a whole. It is important that the data structure used permit the expression of temporal relationships between data items.

The Office Document Architecture (ODA) effort within the International Standards Organization is an attempt to provide a standard architecture for typical office documents, which are mixtures of textual and graphical information. Such documents have a logical structure, a layout structure and a set of logical-logical, logical-layout, and layout-layout
relationships. Such documents are single dimensional owing to the restriction of the layout to a single output device.

Multi-media data structures, on the other hand, are multi-dimensional; the information contained in such structures must be displayed on several output devices. In addition to the relationships between data destined for the same output device, the parallel operation of the devices leads to cross-dimensional relationships - their ordering in time.

One possible method of recording the cross-dimensional relationships is incorporated in the multi-media document format used on the ARPA Internetwork. Three fundamentally different attributes of time-ordered control are provided within the document structure:

- simultaneous - the affected data items are presented in parallel
- sequential - the data items are presented serially; an
- independent - the data items can be presented in any order

For truly coordinated replay of related information, such a system forces the sender to subdivide the information into small presentation elements; these presentation elements are grouped together using the simultaneous control attribute. Obviously, making the presentation elements smaller permits more fine-grained control of the synchronization. Unfortunately, it is also necessary to give the receiver some idea of the absolute start-time for each group of presentation elements if the original information flow is to be maintained.

It is expected that the transformers which provide the object interface to the devices can be tuned to break up the input data into meaningful presentation elements. A better method of indicating the temporal relationships between data items would be for the transformer to tag each data item with the time of its arrival. The timestamp can be included as an attribute of the data item in the multi-media document structure. This gives the receiver the ability to reason about the best method for duplicating the original time ordering. It is important that the transformer tag the data item, since that more accurately measures when it happened, as opposed to when it was processed.

5 Sharing of correlated information

Section 4 has described the architectural features necessary to permit a multi-media application entity to determine and record the temporal relationships between data items from its different media. Since the primary purpose of multi-media applications is to facilitate communication, the application entity must be able to transmit the data items, maintaining the temporal relationships, to its peers for replay. Besides enhancing the interaction by correlating the different media, the communication between peers must maintain the interaction style of each involved medium. Several aspects of meeting this requirement are discussed below.

5.1 Delay vs. throughput vs. reliability

Voice communication requires that the communication path provide high throughput, reasonable reliability and low delay. Existing data communication applications fall into two categories: 1) bulk data transfer, which requires high throughput, high reliability and reasonable delay; and 2) virtual terminal emulation, which requires low throughput, high reliability and low delay. Experience with existing systems seems to indicate that applications are forced to live on a surface in the three-dimensional space of (throughput, reliability, delay), i.e., these three dimensions are not independent. The integration of different media by multi-media applications would seem to require that all three coordinates must be optimized at once, a difficult problem if they are not independent.

The requirements along these dimensions can be further quantified. Voice requires not only that the average delay be small, but that the variance of the delay distribution (jitter) must be quite small as well. The throughput requirement of voice implies that the communication channel must provide guaranteed bandwidth, i.e., not just high bandwidth, but one exceeding a given threshold at all times. It is to satisfy these requirements that the use of circuit-switching techniques for carrying voice has been introduced.

Suppose that an application chooses to transport the correlated presentation elements of §4.2 over a single communication channel. (The possibility of sending different media portions of the presentation elements over separate channels, with each channel optimized for the specific media type, is discussed in §5.2 below.) The application must be able to solicit guarantees from the network and the processor such that the delay requirements of voice (mean and variance of the delay distribution are within prescribed bounds) and the reliability requirements of data (all associated data must arrive reliably) must be met. If voice/video communications are being sent in real time, it will be extremely difficult to find a communication medium which can meet these guarantees.
5.2 Synchronization

To overcome these problems, we could choose to send each media type over individual channels optimized for the particular media. In this situation, one is then confronted with the problem of resynchronizing the different media at the receiving end. Such a solution is viable only in situations where the average, reliable transmission bandwidth of the data communications channel matches (or exceeds) the actual average bandwidth required. Timing information (with respect to the clocked channel) has to be relayed to the receiver, possibly over a separate channel, permitting the receiver to orchestrate the synchronized replay. The action to take if a data item arrives late is application-dependent.

Even if each channel is guaranteed to deliver its data with no variance in the delay distribution, a constant mean delay may be introduced by the technology used to implement the channel. For example, data may be sent over a high-bandwidth satellite channel, while voice is sent over terrestrial circuits. Even if the data packets are transmitted without error, an average delay of about 1s will be introduced. To provide the most generality, it should be possible for the receiver to adjust the amount of skew it is willing to tolerate between related channels. There may be situations where the lack of temporal synchronization is permitted by the semantics of the application.

5.3 Interactivity

One possible way of characterizing the communication between peers is by its interactivity, i.e. the degree of real-time interaction which takes place between communicants. One can intuitively give examples of the extremes for this metric: a normal telephone conversation, with essentially instantaneous peer-to-peer communication, exhibits high interactivity; and a TELEX message, with no communication by the recipient, displays low interactivity. If an application is able to vary its interactivity, it will be able to adapt to the communication environment in which it is embedded. Before describing the architecture's ability to support such adaptability, two points must be established:

- do intermediate points exist in the interactivity spectrum
- can the interactivity metric be quantified?

5.3.1 Existence proof of intermediate interactivity

Consider a telephone call between one person in Europe and another in North America. Sometimes the circuit is provided by transatlantic cable, resulting in normal telephone service; at other times, the circuit is provided by geosynchronous satellite, resulting in end-to-end delays of the order of 1s. In this latter situation, collisions detected by the communicants during initial attempts to use the satellite circuit as if it were a normal voice circuit cause them to adapt their mode of communication. In the worst case, the absolute requirement for normal conversation capabilities leads to one of the participants terminating the call. More frequently, the end-to-end delay causes the conversation to lapse into traffic patterns characteristic of a broadcast channel, i.e. one of the communicants owns the channel, and explicitly transfers the channel ownership as part of the information he is currently broadcasting. The conversation naturally adapts to alternate transmission of voicegrams, where the size/duration of each voicegram is conversation-specific.

In spite of the full-duplex communication channel which is provided by a telephone circuit, an effective telephone conversation is an inherently half-duplex operation. The lack of associated visual cues leads most individuals to signal release of the channel by an extended period of silence. This silent period should cause the listener to begin responding; the lack of a response after this period of time will often cause the transmitter to question the integrity of the circuit. A lifetime of conditioning has resulted in a standard period of silence, based upon the transmission characteristics of circuits carried by terrestrial cables.

Alternatively, one can say that normal telephone conversations inherently consist of voicegrams. The small delays associated with terrestrial circuits have resulted in an implicit broadcast channel algorithm based upon timeouts. Voice circuits over satellite channels operate in a region of much larger delays. The timeouts which are natural in the short-delay regime are no longer appropriate. When timeouts appropriate to the long-delay situation are adopted, the voicegram nature of the conversation becomes more apparent.

5.3.2 Quantification of interactivity

The preceding section has shown that the delays introduced into a telephone conversation by satellite transmission lead the user of the medium to adopt an intermediate form of interactivity. As the delay in the circuit increases, the amount of information packed into a voicegram becomes larger. At one extreme, one finds normal terrestrial telephone conversations, where it appears that the voicegram size is essentially zero; at the other extreme, the "voicegram" is an entire TELEX message. The inverse of the voicegram size is then a suitable
definition for interactivity, the interactivity of the
TELEX message is 0, while the interactivity of a
normal telephone conversation is ∞.

5.3.3 Interactivity supported

The architecture which has been described in the
previous sections supports the notion of interactivity
as a matter of course. The packetizing performed by
the transformers provides an upper bound on the
level of interactivity possible: if an application
chooses to pack several of these presentation
elements into a larger structure before
communicating the data to its peers, then the
interactivity is reduced. The application is free to
select the appropriate amount of packing to perform
based upon application semantics and available
transmission bandwidth. Transmission of the
accumulated data structure might occur, for
example, due to a cue (user pushes a button), the
expiration of a timer or the result of an adaptive
growth algorithm (silence detection). Regardless of the
method used, the primary goal for the application is
to maximize the interactivity subject to bandwidth
and semantic constraints.

6 Examples

Three example applications of the architecture,
spanning the interactivity metric, are provided in
this section. While not exhaustive, they do show
how the architecture can be used to design very
different types of multi-media applications.

6.1 Multi-media mail

This is by far the simplest example, since it requires
an interactivity of 0. A mail composition application
simply packs each datum from each medium into the
multi-media document structure until the user has
completed the creation of a message. The resulting
structure can then be sent as a document using the
X.400 2 series of protocols. Note that the
synchronization support provided by the
architecture permits the exact temporal
relationships between the data items to be reflected
in the document structure. The eventual replay of
the message can be as realistic as the receiving
workstation permits.

6.2 Electronic blackboard

Imagine an application which provides the
appearance of a shared blackboard between two (or
more) workstations. A chalk token is used to
indicate who is drawing on the blackboard at a given
time. A voice channel between the workstations is
also active, to permit voice annotation. As the token
holder draws on the blackboard, the drawing appears
on each screen. Any voice annotation is replayed
synchronously with the graphic elements to which it
is related.

Assuming that the application has access to a
reliable channel which provides more than enough
bandwidth for the graphics updates, the telephones
can be interconnected through the normal circuit
switched telephone network. Each graphics update
is then tagged with a time relative to the completion
of the telephone connection to permit the receiving
application to coordinate the replay of the graphics
with the received voice.

If a reliable channel of sufficient bandwidth is not
available, the application can choose to send the
voice and data over a single channel. The expected
sparse nature of the voice annotation (individuals
seldom talk continuously while drawing) means that
the high interactivity of the telephone circuit is not
necessary. The bursts of voice can be packed
together with the graphics updates to which they are
related, and the application can choose when to send
the structure to its peers according to a variety of
considerations:

- the bandwidth available for communication with
  its peers
- the use of a button
- the detection of an extended period of silence,
  indicating the end of a phrase
- regular timing signals as provided by a local clock

As described in §5.3.2 above, the application can
choose to pack more information into the
accumulated structure (by increasing the period of
silence which indicates the end of a phrase, or by
increasing the interval between timer signals) to
match the communication bandwidth available. Of
course, the interactivity of the application decreases
as it adapts in this manner.

6.3 Military command and control

Military command and control systems are typified
by an intolerance for delay. It is always more
important to receive the most recent data in a timely
fashion than to receive stale data reliably. The
interactivity required of such applications must not
only be maximized, but must always exceed a given
threshold for the application to have any utility.

Consider a submarine task force, consisting of a
group of destroyers equipped with sonar equipment.
The local command post in each destroyer consists of
a display screen equipped with a light pen which is
connected to a computer system. The computer
system is also connected to broadcast communication
facilities interconnecting the ships of the task force. The computer system on each ship constantly updates the display with the most recent sonar image. If the sonar technician detects any unusual activity, a region in the vicinity of the activity can be selected using the light pen. The sonar information for that region, in addition to being displayed locally, is forwarded to a central command post (in real time), possibly with voice annotation; a coordinator at the central command post monitors such information and issues directives for action.

The extreme interactivity required has the following implications:

- the transformers associated with the various input devices will be tuned to provide the smallest granules of information (highest possible interactivity)
- unusual activity presentation elements, which consist of graphics and voice data items, will be sent via separate channels, with no attempt to synchronize the channels (no loss of interactivity due to delay)

7 Discussion

Probably the major reason that multi-media applications have remained so intractable is the severe set of guarantees they demand of their supporting environment. These demands primarily concern two types of resources:

- network bandwidth - real-time transmission of voice/video forces the dedication of enough network bandwidth to handle the worst case, whereas, data traffic is usually handled by dedicating enough bandwidth to handle the average case
- processing bandwidth - the clocked nature of real-time voice/video connections requires that the application be able to schedule itself to meet the timing requirements necessary to service these connections

The current efforts assume that these guarantees have been met. Little can be done if this is not the case.

The greatest single requirement for the support of generic multi-media applications is a mechanism which permits an application to determine and transmit the temporal relationships between the data items from different media. This requirement impacts the scheduling of the application; it also forces document architectures that wish to embrace multi-media systems to provide for this temporal information.

Since multi-media applications are meant to enhance communication between peers, they must be able to adapt to the communication and processing environments in which they are embedded. The ultimate goal of such an application is to maximize interactivity, while maintaining critical temporal constraints.

There is a substantial amount of effort in the area of physical-level integration. The primary motivation for these efforts is to amortize the cabling plant costs of an organization over video, voice, and data services. None of these efforts truly impact the problems of service-level integration, as the physically integrated media provide the illusion of separate communication channels with the appropriate characteristics. The machinery described in §3.5 is still needed, in its entirety.

The architecture presented in this paper is a first step towards a standard way of looking at these real-time applications. As with any architecture, its acceptance will depend upon its ability to describe the applications in this category (generality), while capturing enough of the basic structure to aid implementors in their efforts (specificity). We are currently using the architecture as a basis for prototype development to test its specificity.

8 References


