# Handling Audio and Video Streams in a Distributed Environment

*Alan Jones*[†], *Andrew Hopper*[†‡]

[†] Olivetti Research Ltd.
24A Trumpington Street
Cambridge CB2 1QA
United Kingdom

[‡] University of Cambridge Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
United Kingdom

## Abstract

Handling audio and video in a digital environment requires timely delivery of data. This paper describes the principles adopted in the design of the Pandora networked multi-media system. They attempt to give the user the best possible service while dealing with error and overload conditions.

Pandora uses a sub-system to handle the multi-media peripherals. It uses transputers and associated Occam code to implement the time critical functions. Stream implementation is based on self-contained segments of data containing information for delivery, synchronisation and error recovery. Decoupling buffers are used to allow concurrent operation of multiple processing elements. Clawback buffers are used to resynchronise streams at their destinations with minimum latency.

The system has proved robust in normal use, under overload, and in the presence of errors. It has been in use for a number of years.

The principles involved in this design are now being used in the development of two complementary systems. One approach explodes Pandora by having the camera, microphone, speaker and display as independent units linked only by the LAN. The other approach integrates these devices as peripheral cards in a powerful workstation.

## 1.0 Introduction

The Pandora project demonstrates the practicability of bringing real-time audio and video traffic onto the desktop via a general purpose ATM network [Hopper90]. The data is digitised at source, transmitted using standard protocols [McAuley90], and returned to analog form at the eventual destinations. The Pandora's box [King92] provides all the real-time functions and the ATM network connection, while the applications run under the X Window System on a stock workstation. About 50 Pandora boxes have been built since they were first deployed in February 1990, and many continue in use around the offices of Olivetti Research Laboratory, the University of Cambridge Computer Laboratory and Engineering Department, and University College London.

### 1.1 Pandora Hardware

The Pandora's Box is a hardware sub-system that can be controlled by a workstation over a 10Mbit serial line (an Inmos Link [Inmos89]).

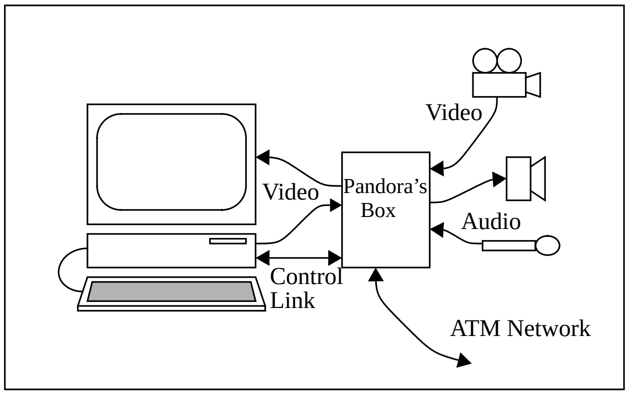

Figure 1.1 Pandora Connections

It intercepts the analog video output from the workstation, and can insert areas of active video from its own framestore under the control of a mask. It can also digitise video from a camera, and it can receive and transmit streams of data over a dedicated ATM network connection [Hopper88]. An audio board within Pandora's box can accept microphone or telephone input, and can drive loudspeakers or telephone handsets. The box is controlled by commands from the host workstation that instruct it where to send video from the local camera, where video windows are to be placed, which parts of them are visible, etc. The host states what it wants done with the streams, and they then run continuously until stopped.

There are five boards in the box as shown in figure 1.2.

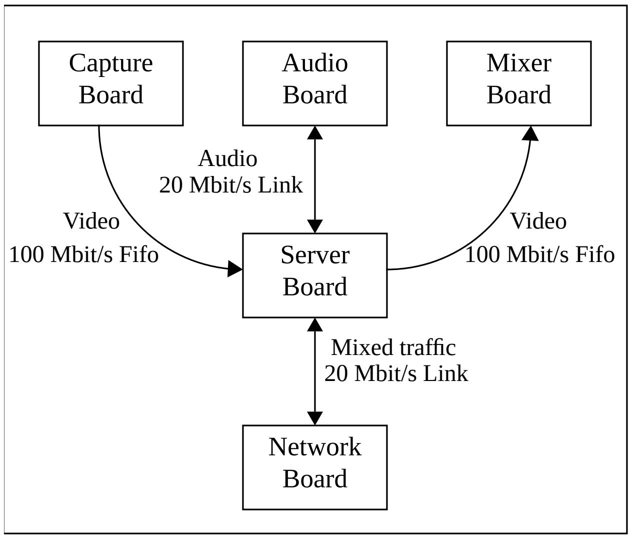

Figure 1.2 Data Transport Between Transputers

| | |
|---|---|
| **capture board** | Digitises camera input |
| **mixer board** | Assembles video into a framestore and mixes it with host output |
| **audio board** | Transports audio streams to/from 8kHz codec |
| **server board** | Acts as a data switch between the other boards |
| **network board** | Interface to ATM Network |

Each of the boards contains an Inmos Transputer [Inmos89] as a programmable embedded controller. Each Transputer runs common Pandora communication and synchronisation software written in the programming language Occam 2 [Inmos88]. This software communicates over Inmos links, with some high bandwidth data paths provided by memory mapped fifos. The mesh of control links is shown in figure 1.3. Many early pandora's were built without the network card, the server transputer handling the network interface hardware directly. Other early

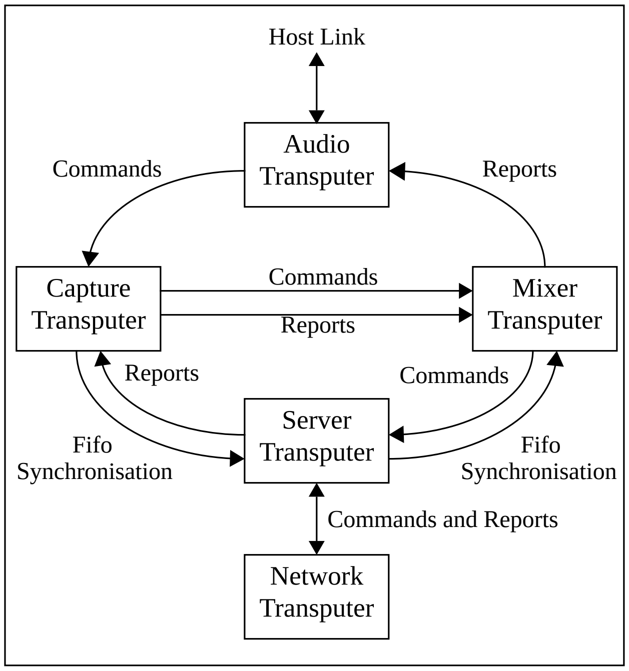

Figure 1.3 Use of the Transputer Links for Control

configurations have included the use of fifos for the transport of audio to the server, a SCSI interface for the host link to the audio transputer, and SCSI hardware on the server transputer for local disk storage of streams..

Almost every process in the Pandora system can receive *commands* and generate *reports*:

- *Commands* are used to set up the operations performed by each process running in the transputer, usually with reference to a stream number that identifies a particular audio or video data stream. To set data flowing, it is necessary to allocate a new stream number, inform each process from the destination back to the source what is to be done to that stream, and then command the source to begin producing data. The data will then flow indefinitely without any further interaction with the host.
- *Reports* are collected from all main processes, and multiplexed together. They are usually in the form of text messages generated when Pandora is overloaded, when some error has been detected, when a command has requested some information, or on occasion just to say that everything is all right. Reports are sent to the host computer for display or logging. They are occasionally intercepted by other pandora processes for the purpose of extracting information that is not time-critical.

## 2.0 Principles

One of the design goals of Pandora was to allow many video windows to be active on the screen at once. Their accompanying audio streams are mixed by software in real-time on the destination transputer. No limit is placed on the number of incoming streams that can be mixed, save that imposed by system bandwidths and CPU resources.

Several desirable attributes of the final system were identified in the form of principles which were used to guide the design. They have different implementation costs and importance in different parts of the system, and should be considered wherever they might be relevant. The principles are described here under the headings: priorities, splitting, and timing.

### 2.1 Priorities

The first four principles deal with priorities between streams, and are mostly used where there is a choice of data streams to process next. By default, we share resources evenly, but sometimes find that better results can be obtained when some bias is given in the directions suggested by these priorities.

**Principle 1: Outgoing Priority**

The Pandora software is designed to be operated under conditions of slight overload - users are observed to increase the load on such systems until they just begin to break. To accommodate this very natural behaviour, it is desirable that the user whose box is overloaded is the one who first observes the degradation in quality so that the correct action is more likely to be taken. This leads us to a principle that guides the implementation:

*Under overload, incoming data streams should be degraded before outgoing data streams.*

This principle applies amongst Pandora's Boxes, but is reversed for repositories, where the incoming data streams should be recorded as accurately as possible, even if that means degrading streams that are currently being played out. It is a simple matter to play a stream again, but recording one again could present greater difficulties.

**Principle 2: Audio Priority**

When the system becomes overloaded, the users ought to be able to communicate as well as possible so that they can make sense of what is happening and put it right. In most cases, overload conditions cause loss of data on some streams. It is preferable to lose some or all of the video data before the audio is degraded. Efficient communication is difficult with video only or with a noisy audio stream. Again, a principle can be defined:

*Under overload, video data streams should be degraded before audio data streams.*

**Principle 3: New Stream Priority**

In a multi-stream system the user does not have to shut down one stream before opening another. For instance, a video call may come in while several other streams are being displayed (local video, a television program etc.). Although this may overload the system, the user should be allowed to open the new stream, observe the degradation, and decide if it is worth shutting something down. To make this more convenient for the user, we define another principle:

*Under overload, data streams that have been open the longest should be degraded first.*

In the example of the unexpected video call, the new streams will be given priority, so the user may not have to bother closing the other streams unless they are a nuisance in some other way. When the video call is terminated, the other streams will return to full bandwidth and no data loss.

**Principle 4: Command priority**

In a real-time system, it is tempting to deal with the time-critical data streams first, and process new commands in otherwise idle time. However, if the time-critical tasks exceed the system capacity, then it is important that commands are still passed to, and executed by, their destination processes, otherwise it might not be possible to bring the system back to normal.

*It should not be possible for stream processing to prevent the transport and execution of commands.*

### 2.2 Splitting

**Principle 5: Upstream independence**

The data streams from Pandora are often copied to several destinations (stream splitting). It is desirable that the bad performance of one destination due to overload should not affect the recipients of other copies of the data. We implement this, at the points where the data is duplicated, by not sending a segment if the next process down the line to a particular destination is not ready. It is then the destination's responsibility to detect (by segment sequence number) and recover from this. Here is the principle:

*Downstream performance bottlenecks should not affect streams that have been split off earlier.*

**Principle 6: Continuity during reconfiguration**

Consider the case where we are recording a stream when another destination is added to it. It would be annoying if there were a break in the recorded data when this happened, so the software is designed to allow the plumbing to change while the data flows through undisturbed.

*Splitting a stream to an extra destination, or closing down one of several destinations, should not affect the other copies of that stream.*

### 2.3 Timing

**Principle 7: Minimise delay**

Audio performance is critical to the perceived quality of a system. In our case, the most challenging application was the hands-free video conversation, since there is a muted echo back from the far end of any conversation. If this echo is delayed more than a few tens of milliseconds, the effect is quite irritating. It is also irritating if the video lags appreciably behind the audio. In the real world, we are used to seeing events slightly before we hear them.

*Delays in the data streams must be kept to a minimum at all stages.*

**Principle 8: Local adaptation**

It is difficult to predict the bandwidths and critical times within a system where many different combinations of audio, video and data streams could be in use. To comply with some of the above principles, decisions have to be made about how much data to buffer, what time to start throwing data away etc. A further principle guided the decision code:

*If possible, timing and buffering decisions should be adaptive to current, locally observed conditions.*

This design style also allowed the code to take advantage of hardware and software improvements without laborious retuning, and thus encouraged experimentation.

## 3.0 Implementation

In the design of the real-time software, we found that the most severe constraints were imposed by the audio streams. The ambience of a two-way live conversation is very sensitive to audio delays in the 10 to 20ms range, whereas video has more of a supporting role and can stand delays in the 40 to 80ms range before conversation becomes impaired. Furthermore, errors in the continuity of an audio stream are much more noticeable than in a video stream.

### 3.1 Overview of the Inmos Transputer

The transputer is designed to allow very low overhead context switches. This is achieved by having no cache, memory mapping or registers, but direct hardware implementation of process structures, scheduling and interprocess communication mechanisms. The result is a CPU that can, if carefully programmed, do useful amounts of work even if context switches are taking place at rates greater than 100kHz. The CPU also includes a timer register with a resolution of one microsecond that can be used to schedule tasks very precisely.

Interprocess communication is by rendezvous between the sender and receiver of some data on a unidirectional transputer *channel*. The hardware scheduler will automatically block the first of the processes (sender or receiver) to reach the transfer (output or input respectively) on the channel, and bring it back onto the run queue when the rendezvous is complete. Processes can be waiting for inputs, including timer expressions, on several channels at once (see the ALT construct in Occam 2). The alternatives in the clause can be prioritised so that important channels (such as those receiving commands) cannot be ignored even if other alternatives are always ready.

Each transputer includes four high speed (5, 10 or 20Mbit/s) serial lines called Links. These can be connected as point-to-point communication channels between transputers. They use DMA to transfer data from the memory of one transputer to that of another under the control of the same channel read/write commands that are used for on-chip rendezvous, and with similar synchronisation. Links are supported directly by the hardware, so that the time that elapses from the presentation of data by the sending process to the resumption of the receiving process on another transputer can be as little as a few microseconds.

The transputer has an event pin which acts like another link input, allowing the CPU to be interrupted according to external conditions. A high priority process waiting (descheduled) for the event pin can be running in less than 1 microsecond.

### 3.2 Audio Format

Audio is sampled by a standard 8-bit μ-law codec at 125μs intervals. It is handled in blocks of 16 samples, representing 2ms of audio. For the purposes of transmission outside the audio board, a number of these blocks are grouped together with a header to form a pandora segment (figure 3.1).

Each field in the header is 32 bits in length. The first five fields, up to and including the length, are common to both audio and video segment formats.The segment header completely describes the audio samples that follow. It contains in encoded form the format used, sampling rate, sequence number and carries a timestamp with 64μs resolution derived from the Transputer clock as close as possible to the data source.

The timestamps are relative to the last time the Pandora's Box was booted, and are not drift corrected. They are not necessary for the operation of any of the Pandora's Box

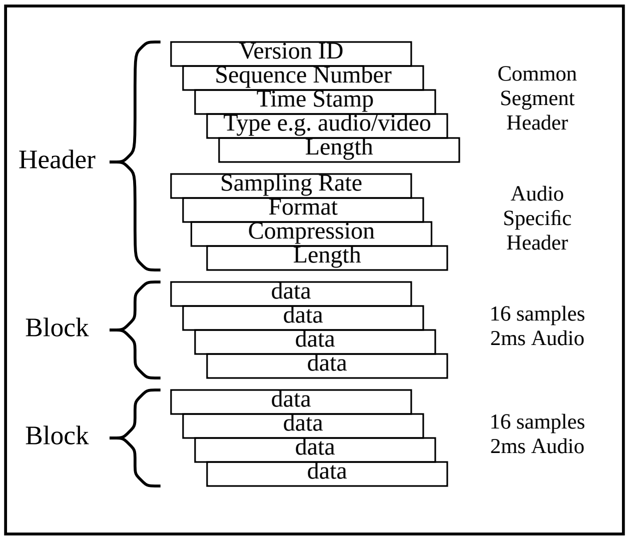

Figure 3.1 Pandora Segment Format for Audio

code, but are included for fault diagnosis, and for the convenience of other systems working with Pandora such as the Repository. They will be synchronised to a global time standard: GPS time [Nato91], as soon as the time service becomes available on our ATM network. This will release us from the present requirement that streams to be synchronised during playback must have been recorded on the same repository, where their timestamp offsets are recorded.

The number of blocks in each outgoing segment can be varied. We usually run with 2 blocks per segment (principle 7), but can alter this dynamically if the recipient cannot handle the arrival rate (perhaps using 12 blocks = 24ms) or if we want a particularly low latency (1 block = 2ms). Incoming segments of any mixture of sizes are accepted.

A major use of this facility is when streams are stored on a repository. As they are no longer live, there is no requirement for low latency, and we would like to reduce the disk space taken up by headers. This is done as a separate operation after the stream has been recorded, by splitting out the 2ms blocks, and merging them to form 40ms long segments containing 320 bytes of data plus a new 36 byte header. These can be played back directly to any Pandora box.

## 3.3 Video Format

A stream of video comes from an arbitrary rectangle of the field of view of the camera. The capture transputer can read several streams from different overlapping rectangles.

Video segments do not have to contain a whole frame. A frame can be broken up into a number of rectangular segments, so the segment header contains a count of the number of segments in the frame, the number of this segment within the frame, and enough information to place this segment in the correct position (figure 3.2).We have a variable number of fields after the compression type field so that compression parameters for any scheme can be accommodated. Compression schemes and parameters can be changed from one segment to the next to give the fastest possible response to user requirements.

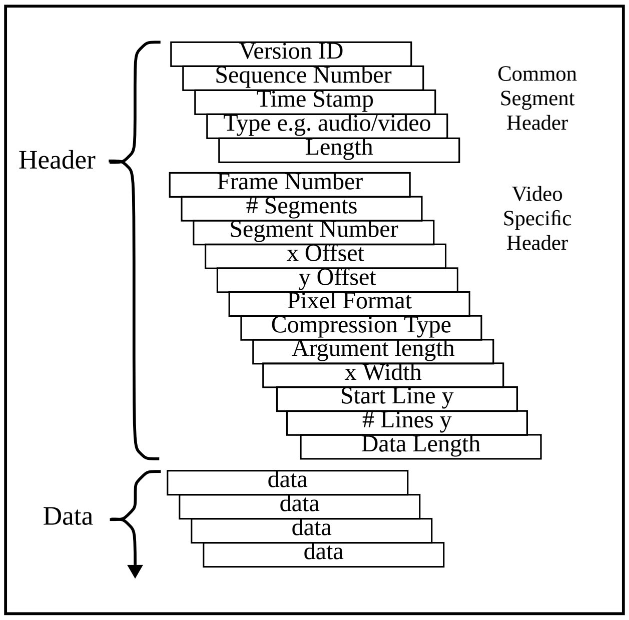

Figure 3.2 Pandora Segment Format for Video

## 3.4 Server Data Transport

On the Server Transputer, through which all Pandora data streams in a box must pass, it is clear that unnecessary copying of data is to be avoided. It is possible to copy data straight from its source (fifo or link) to its destination using a single block move instruction. However, this would not work well for the following reasons:

- In the case where there are multiple destinations (e.g video to be displayed both on the local screen and sent out over the network) we would have to copy into local memory (evaluation registers or on-chip memory) in order to perform the duplication.

- Synchronisation of such transfers would break the stream independence principle. A delay waiting for the destination for stream A to be ready would hold up stream B coming from the same input device, even if it is travelling to a different destination and could proceed immediately.

For these reasons, we chose to copy once into memory, and once out for each output device that wants the stream. The input processes obtain empty buffers from an allocator process in advance, fill them as the data become available, and then transmit the buffer index numbers through the rest of the system. Figure 3.3 shows the main datapaths through the Server Transputer.

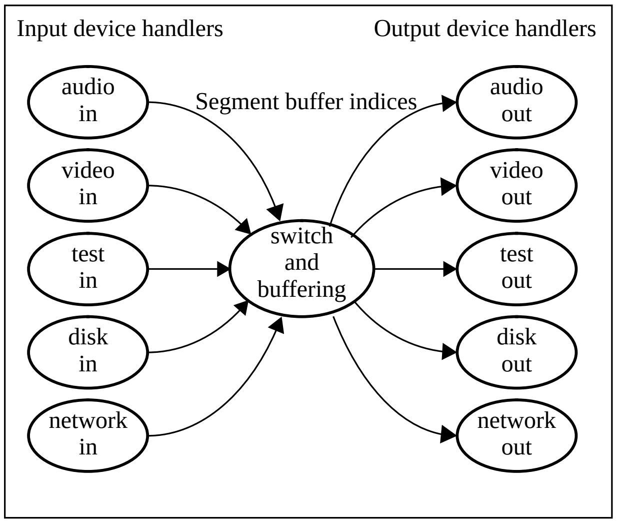

Figure 3.3 Overview of Data Paths in the Server

Each input device handler is responsible for receiving pandora segments encoded over links, fifo's, software generators, SCSI interface or the ATM ring network. Similarly, the output device handlers are responsible for different classes of transmission hardware. Each handler is a separate Occam process that runs for all time, and handles many simultaneous streams of buffer indices. Each of these top level processes are addressable objects that can be configured or interrogated from the host, network or internal Pandora software via their command and report channels.

It should be noted that Occam processes are built by sequential or parallel composition of simpler Occam processes, right down to the individual assignment or I/O processes that are the statements of the Occam language. These diagrams can only show certain levels of the process structure. Looked at in detail, many of these processes will be found to contain several long-lived Occam processes inside, and to interact with other service processes such as command and report routers. Some of these internal processes do not terminate, while others have lifetimes measured in microseconds.

Figure 3.4 shows a little more detail of a typical datapath from the audio input handler to the audio output, complete with the Occam channels that are used to interact with the allocator process.

The buffer memory is shared by all the processes that may use it. The allocator keeps a reference count of the number of processes using each buffer, and so must be informed whenever:

- A process has finished with a buffer without passing it on (to decrement the reference count).

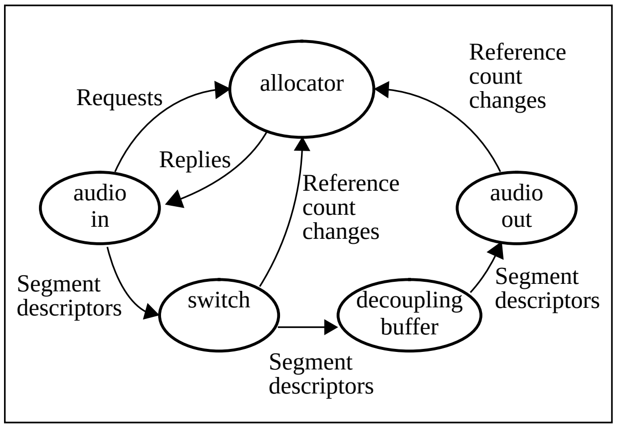

Figure 3.4 Example Buffer Management Channels

- A buffer descriptor has been sent to more than one other process (to increment the reference count).

The common case of a process passing on a descriptor to just one other process does not require a change in the reference count.

If there are no buffers available, then the allocator will not listen for any requests, and the requesting processes will be descheduled by the usual channel synchronisation mechanism until the allocator is ready to receive again. The allocator reports this (serious) fault on its report channel so that it can be logged.

All the separate streams of data through the Pandora Box are labelled by stream numbers allocated by the interface code [Wray89]. Incoming streams from the network carry the stream number allocated by the destination box in their VCIs (Virtual Circuit Identifiers used by the ATM network), and streams within pandora pass the stream number in an extra field preceding the segment header.

Any process which handles a variety of streams in differing manners will use the stream number to index private tables that describe the operations to be performed on the segments of each stream (e.g which processes to send them to, what outgoing VCI to use etc.) and hold the state of that stream (e.g. number of dropped segments, muting level etc.). The tables are updated without disturbing the flows of data when commands are received from the interface code, (principle 6). A command will be received as soon as the process has finished dealing with any current segment, so that there is no possibility of the table changing during the processing of a segment, nor of high data rates locking out commands (principle 4).

## 3.5 Audio Data Transport and Buffer Allocation

The 125µs samples from the codec are written continuously into a byte-wide fifo. Every 2ms, the Transputer event pin is signalled, and the code notes that another 16 bytes (a block) are in the fifo (figure 3.5).

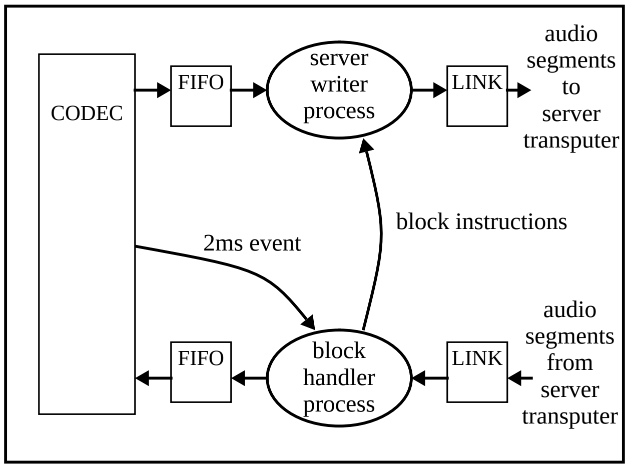

Figure 3.5 Movement of Data on the Audio Transputer

When sufficient 2ms blocks have accumulated to justify the overhead of a Pandora segment header, the server writer process is ordered by the block handler to transmit them to the server board. The server writer is implemented as a separate process to allow some concurrency in case the Server is busy, or in case the CPU is needed urgently for other block handler tasks such as audio mixing.

The data transmitted over the links are placed in transputer memory, and then the appropriate process waiting for input from that link will be set runnable by the transputer scheduler. If a process writes to a link before the previous message has been received by the recipient, then the writer will be blocked until the first communication is complete. This mechanism is implemented by the transputer hardware, and is used for interprocess synchronisation throughout Pandora. A context switch can be accomplished in less than 1μs, so we are not imposing too much of an overhead when we add another process into the chain.

## 3.6 Video Data Transport

Rectangular blocks are read from a video framestore at intervals determined by the requested frame rates of the streams. Each stream can be from different, possibly overlapping, sections of the store. The frame rates are expressed as a fraction of full 25Hz frame rate. For example, 2/5 gives an average of 10 frames per second. The reading of the blocks is carefully timed so that the data from the camera being written continuously on a second port does not update any part of a block while it is being read.

Large blocks may be sent as several pandora segments, each of which is despatched as soon as the data is ready, reducing latencies and buffering requirements. The segment headers are sent over the transputer link to the server, while the data goes via a fifo to a compression subsystem.

Each segment of video data is reduced further into several slices of a few lines each for transmission through the compression subsystem. After each slice has been written to the fifo, a small description containing the number of lines and their length after compression is sent over a link to the server transputer. Each line of video data has a one byte compression header added, which is used by the compression hardware to determine what sub-sampling and DPCM coding should be applied. When the last slice has been sent, a tail marker is sent over the link. A header slice description precedes the first slice of a segment to describe what compression algorithm has been selected, what stream number the segment is for, and contains the full segment header.

The slice descriptions on the link can be considered to be a model of the data that is in transit through the fifo's and compression hardware. We buffer a few slice description in software to allow several slices to be in transit at a time and so increase concurrency across the link. This is especially useful because our compression hardware is pipelined and does not drain automatically. In order to flush the last slice of data from the pipeline without waiting for the next segment to arrive, we send a few dummy lines after each video segment. The server transputer must not attempt to read these dummy lines until some other data has pushed them through the compression engine, so one of the buffers on the link is special. It is designed to always hold back one slice description at all times, with any tail or head descriptions that follow, until another slice description is read. In this way, the buffer chain models the slice of data that will always be held in the compression pipeline, but still allows for several slices to be in transit when necessary.

The batches of video data are assembled into a segment buffer on the server transputer as they arrive, and its index is launched into the rest of the system as soon as the tail marker is read from the link.

The route through the decompression hardware from the server transputer to the mixer transputer is similar to the compression route. The decompression hardware expands the DPCM coded video, and can also interpolate both horizontally and vertically. A problem arises when we interleave segments from different video streams, as the vertical interpolation for the first line of a segment needs to know what the last line of the previous segment contained. We had three choices:

1) The interpolation hardware keeps a copy of the last line processed on each stream - too costly.

2) Each segment contains a copy of the required lines from preceding segments - ties segment format too tightly to the particular hardware in use.

3) Maintain a software cache of the last line processed on each stream, and reload the interpolation hardware whenever we interleave segments. This is what we chose to do.

This is just one example where the requirement of multiple low latency streams has an impact on the design of hardware components. Many commercial components and systems are being designed with single stream operation in mind, with little thought being given to the requirements of multi-stream environments such as Pandora.

On the mixer board, the video data is copied from the fifo into a waiting memory buffer. We do not display any part of a video frame until all of the segments have been received, otherwise the effect of a tear can be seen when part of the image is moving parallel to a segment boundary. Once we have all the data for a frame, it is copied into the display frame buffer as soon as possible, care being taken to avoid the scan of the display controller, as this can also lead to tears. The ability to schedule processes with precisions of a few microseconds allows us to make full use of our knowledge of the display scan, copying frames both in front of and behind the scan if necessary.

## 3.7 Buffer Queues

There are two sorts of buffer queue used in the Pandora audio paths, *decoupling buffers* between processes or hardware units that do not run synchronously, and *clawback buffers* used to enable streams with jitter to be synchronised with the local clock (figure 3.5). Each buffer queue is implemented by a separate Occam process.

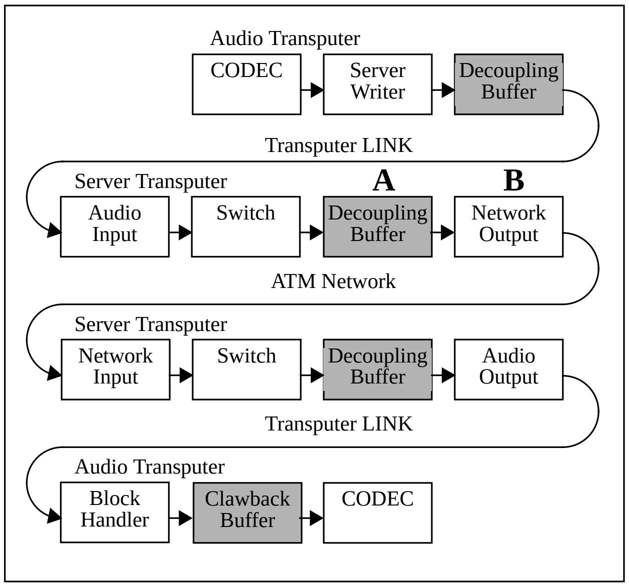

Figure 3.5 Movement of Audio Data between Boxes

### 3.7.1 Decoupling buffers

These are generic circular buffers, holding a FIFO queue of references to pandora segments. In addition to an input and an output channel for segment references, they also respond to commands and generate reports. Some are used for audio only, while others are buffering segments of all types.

If the buffer is full, the process will not be listening on its input channel, so another process which attempts to send on that channel will be blocked until an item has been read from the buffer. If such behaviour is undesirable (principle 5), a form of the decoupling buffer with a *ready channel* for the input can be used (figure 3.6).

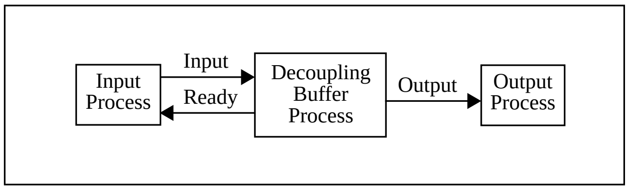

Figure 3.6 The use of a Ready Channel

After an item has been accepted on the input, if there are still more free slots, then the ready channel signals TRUE immediately. If there are no more free slots, then it signals FALSE immediately, and TRUE later when a slot becomes free. The input process can then choose to throw away the data rather than block waiting for the buffer to become free.

It is important that the ready channel always sends a reply immediately. Suppose that it simply signalled TRUE when it was ready to receive input again - a conventional acknowledgement. Then it is possible that the decoupling process would be descheduled between receiving some data and sending the acknowledgement, and the upstream input process would not know whether this was due to the buffer being full (which would lead to blocking if it tried to send more), or just scheduling (in which case it can safely send more). To avoid this, the upstream process will block on input from the ready channel until it gets a reply one way or the other before continuing, and the decoupling buffer will send an immediate reply after every input indicating whether or not it has more free buffers (i.e. whether it is READY). After a FALSE reply, the input process will not send any more data on its output to the decoupling buffer, but will listen on the ready channel in addition to its other inputs. When it subsequently receives a TRUE reply on the ready channel, it sets a flag indicating that the corresponding output can be sent data again, and ceases to listen on the ready channel.

The decoupling buffers should not normally become full (principle 7). They are inserted to allow some concurrency between processes or independent hardware units. If they do become full, then it means that 1) the system is overloaded, and 2) the scheduling is such that the input processes are handling more segments than the output processes. In Pandora, we arrange that the output processes have priority, and that the input processes run without data loss as far as the decoupling buffers, at which point they throw away data if the buffers are full. If the transputer has too few CPU cycles to handle the data, then the output processes will take priority, and the input side will be held up, causing back pressure that will allow data to be thrown away closer to the source.

The decoupling buffers on the server transputer are placed downstream of the switch so that the poor performance of one output device does not affect streams to other output devices (principle 5). If an output device falls so far behind the input that its decoupling buffer fills, then the switch simply omits to send it any more segments (effectively discarding traffic for that output only) until the buffer has free slots again. The switch records how many segments have been dropped in this way, and periodically sends reports while the condition persists.

The first limit that tends to be exceeded in normal operation is the bandwidth of the interface to the network. We see this as slow response from the network output process (figure 3.5, mark B), leading to an excess of data held in the decoupling buffer (figure 3.5, mark A) before it. We limit the size of this buffer so that the video delays do not become aggravating to the user, and buffer the audio separately so that it can be given priority (principle 2). See figure 3.7.

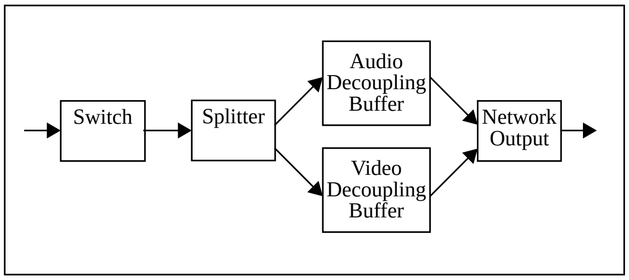

Figure 3.7 Splitting out the Audio Data

The decoupling buffers are attached to command and report channels in the same way as all other Pandora processes. If a stream degrades, a command can be used to request a report from the buffer process, that will include its present length (indicating where any delay is being introduced), size limit and pointer positions (indicating how active it is). It is also possible to specify a new buffer size dynamically, and the buffer will adjust to this size without any loss of data. During debugging, this can be used to see if a buffer has become full because it is too small to cope with the jitter in the input and output processes, or whether there is a consistent overproduction of data which will fill a buffer of any size.

### 3.7.2 Clawback buffers

These buffers are designed to remove the effects of drift and jitter, and should be placed downstream of any components that introduce variable delays. In practice, this means as close to the destination as possible.

The clawback buffers must operate on individual streams, so there is one for each audio stream arriving at the destination (figure 3.8).

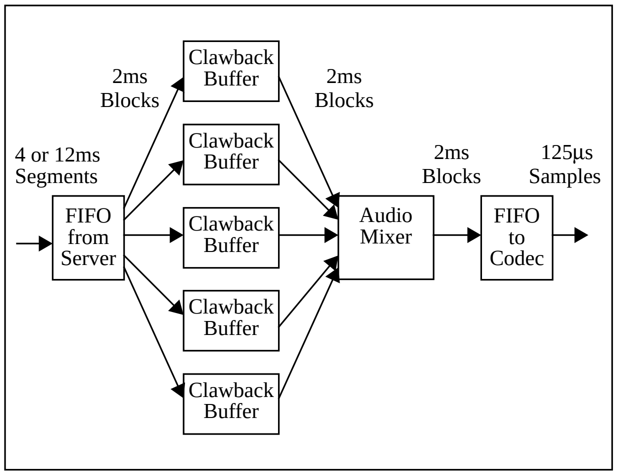

Figure 3.8 Placement of Clawback Buffers

They also tend to have different lengths according to the amount of jitter being removed. So that they can share a common pool of memory dynamically, they are implemented by linked lists rather than circular buffers.

A 2ms block is read from the output end of each buffer every 2ms by the audio mixing code. If the clawback buffer is empty at this time, then it is not included in the mixing. This is equivalent to inserting 2ms of zero amplitude samples into the stream. When the samples do eventually arrive, the buffer will fill to one block more than it would have done, and will hopefully have enough data buffered in future to cover the time waiting for the next late block. If the source clock is slower than the destination clock, then the buffer will always have a tendency to empty, and this mechanism will insert occasional silences appropriately. If the effect of the zero samples were found to be annoying to the listener, then more sophisticated audio processing could be used to bridge the gap in a gentler way.

When blocks arrive, they are placed straight into the clawback buffer, which need have no limit on its size. In Pandora, we have a total of four seconds of clawback buffering shared between all active streams. With our ATM network, there is no point in buffering more than about 120ms of audio for a single stream, as this will only be exceeded by a serious failure somewhere else in the system combined with overenthusiastic buffering. Therefore we throw away samples if the buffer is above its limit when they arrive. We do not expect the jitter correction to approach the limit even when the network is severely loaded, and the process reports this condition so that the cause can be investigated.

Now we have a buffer that will increase its contents in the presence of jitter, until it holds just enough data to cope with the worst peaks. The problem is, the extra delay introduced is mildly annoying to the users of live two-way streams, as they will hear their own echoes as an unnaturally long reverberations, and the conversation can be hampered if one speaker does not realise that another has already started talking. There is not much that can be done to reduce the delay while the jitter continues, but if conditions improve, perhaps because the users have closed down other streams that were causing the jitter, we would like to reduce the delays back to minimum levels (principle 7) without having to be informed by higher level software (principle 8). This is where the clawback mechanism helps.

The first step in reducing the contents of a buffer is to monitor how close it has come to emptying completely. Therefore, every time a block is added, the clawback mechanism checks the count of blocks in the buffer against a lower target (our default is 4ms). If it is above this target level, a count is incremented. When this count exceeds some value (4096 in our implementation, representing 8 seconds), the current incoming block is dropped to reduce the delay. This mechanism allows the delay for jitter correction to be reduced at the rate of 2ms every 8s, or 1 in 4000; this is called the Clawback Rate. With our network, the jitter is usually around 2ms, sometimes rising to 20ms if there are large blocks of video being transmitted through the same network interface. It will take about one minute to adjust to the change from 20ms jitter correction to 4ms. If this correction were faster, there would be a danger that it would be applied during occasional short intervals of low jitter, and lead to unnecessary degradation of the audio stream when the jitter increased again.

The only remaining problem is clock drift where the source clock is faster than the destination clock. This is covered by the same clawback mechanism provided that the clawback rate is greater than the maximum clock drift rate. Since our clocks are controlled by quartz oscillators with a 1 in $10^5$ drift rate, our 1 in 4000 clawback rate is sufficient to satisfy this condition.

If we could not limit the buffering (and hence the maximum jitter) across our network connections to such a small level, then a multi-rate clawback would have to be used. This would involve keeping a running minimum of the buffer contents, and removing blocks at a frequency proportional to that minimum. A suitable algorithm would be to remove a block and reset the counts whenever the product (minimum contents) $\times$ (blocks since last reset) exceeds some level (expressed in block seconds). 20 block seconds would be suitable for our environment. Then if the minimum contents were 10ms, we would be removing a 2ms block every 2000 blocks, or 4 seconds. If the minimum contents were 50ms, then we would remove a 2ms block every 400 blocks, or 0.8 seconds. The block seconds level represents a time constant for the exponential decay of the jitter correction delay. The time to halve the delay when the jitter source is removed is roughly 0.7 times the level that has been set for the product, which would be about 14 seconds for our example.

The time saved when a clawback buffer is found to be empty is used to deactivate the stream, removing the clawback buffer altogether. If a block arrives for a stream that does not have a buffer, a new clawback buffer will be inserted, and mixing will resume. In this way, the audio code does not have to be informed of the creation or deletion of streams; it just adapts to the incoming data.

The clawback mechanism is an extension of the elastic buffer approach used in many systems [Swinehart83], but allows the buffer limit to be much greater as the buffering will not be used except when absolutely necessary. The efficacy of this approach was demonstrated when Pandora was used in trials of a new country-wide acandemic computer network, SuperJanet. Unmodified Pandora's Boxes communicated audio and video successfully under the high jitter conditions of a connection from Cambridge to London involving several networks and protocol conversions.

To avoid problems with clock drift, some systems dump data from their buffers when some critical amount is reached, and attempt to match the clock rates by receiver clock adjustment [Want88, Ades86]. Such adjustments would not scale well to multi-way audio, and buffers could remain occupied when not strictly necessary.

## 3.8 Error Recovery

If an error occurs, such as a decoupling buffer becoming full or data becoming corrupted, the general rule is that the current segment is thrown away. Processes keep local counts of how many segments have been thrown away, and send messages on the report channel as soon as possible subject to a minimum period between reports for any particular sort of error. These messages are brought together on the host computer, and written to a log file. If a stream is corrupted because of data loss, it is possible to look in the log file to find out whether the data is being lost within Pandora, and if so, which process is losing it and why.

The original intention was that software on the host computer would interpret these events, and take action appropriate to the applications involved by increasing compression, decreasing frame rates, or even asking the user to shut something down. As it turned out, this was unnecessary, as the user of the overloaded machine notices the effects, and tends to shut down unwanted applications without further prompting.

As all pandora segments carry sequence numbers, the destination can detect that segments are missing as soon as a later one arrives. Action appropriate to the type of data can then be taken.

In the case of audio, the recovery from lost data should not create unpleasant sound effects. Single byte samples dropped occasionally were undetectable except during solo violin pieces, and then only because they took place at constant intervals. Dropping occasional 2ms blocks was noticeable in most music, but rarely in speech. If 2ms blocks are repeatedly dropped, the speech sounds "gravelly".

When audio samples have to be inserted, occasionally repeating the last byte sample is again virtually undetectable. Replaying the last 2ms block occasionally is perfectly acceptable for speech, and replaying 2ms blocks frequently gives a garbled effect. We replay the last 2ms block, and try to ensure that it does not happen frequently.

## 4.0 Audio Performance

### 4.1 Applications

The audio streams described in this paper have been used in a variety of applications. Live bi-directional streams are used in video phone and multi-way video call systems. Live one-way streams are used for shout (one destination) and tannoy (multiple destinations) commands. Finally, stored audio streams are used for video recording, playback, and videomail applications.

Sometimes the boxes are all on the same network, but more often, some will be on different networks joined by bridges, or distributed over an ATM backbone network. The audio streams have proved to be quite robust in the face of these changes in the network environment. The main problem recently has been a failure of lip synchronisation (early audio, late video) when problems with our experimental networks caused high loss rates for the video segments. In most instances, the audio is uncorrupted, so the participants can describe the situation and work through possible causes (principle 2).

### 4.2 Throughput

The T425 transputer used on the audio board can mix five audio streams in the straightforward case, but only three if we have jitter correction, muting (see next section), an outgoing stream and the interface code running at the same time.

The 20Mbit/s link to the server transputer is not a limiting factor; it would be capable of taking 100 audio streams if we could process them. The context switching rate is probably around 5kHz, and is not a problem for the transputer.

In the network code, the overheads for processing a block of data must be kept fairly low since audio segments are small blocks, and should not be batched together as this would increase latency (principle 7). Our network code introduces more latency than necessary because segment transmissions are not interleaved. Thus video segments can hold up following audio segments, introducing up to 20ms of jitter in a stream.

When other streams are quiet, the best one-way trip time from microphone input of one box to speaker output of another box over the network was 8ms. 4ms of this can be accounted for in the buffering *to* the codec, and 2ms in the buffering *from* the codec.

### 4.3 Muting

When audio is played hands-free at both ends of a conversation, there is a potential problem with feedback. Carefully positioned loudspeakers and microphones with flat frequency responses can avoid howl-round so long as the loop gain is kept down (that is, microphone gains and speaker volumes are not too high); alternatively, a frequency shifter can be used. However, the residual echo is still distracting, especially if it is appreciably delayed, and the problem becomes worse if several offices are all linked in a conference.

Echo cancellation would be possible in some environments by measuring the frequency (or impulse) response of the room, predicting the received echo given the loudspeaker outputs, and subtracting this from the microphone input. However, the audio transputer does not have sufficient spare CPU cycles to perform this in real-time, so a simpler muting scheme was implemented. This scheme can also be used for audio handled by a general purpose workstation provided the timing constraints of a few milliseconds can be met.

The data stream to the loudspeaker is monitored for samples exceeding a threshold level. When the level is exceeded, the data stream from the microphone is muted in two stages, and returned to full volume after a sufficient time for any room reverberations to die away. The return to full volume only occurs after the loudspeaker output has remained below the threshold for sufficient time. The threshold, muting factors and delay times are all dynamically alterable, but our default values are shown in figure 4.1.

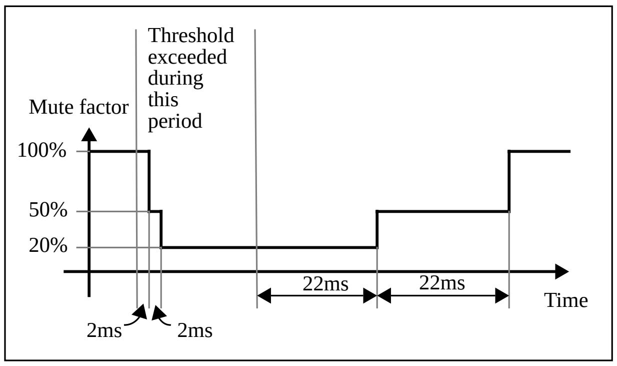

Figure 4.1 Muting Function

The muting is performed by lookup tables that directly scale the 8-bit µ-law samples as they are copied from the codec fifo to the server link.

The two-stage muting was chosen because the steps are not so high that audible clicks are heard when it is applied. The 2ms granularity was chosen for convenience as this is the smallest unit of data that we move around in the audio code. The 22ms at a factor of 20% was chosen so that the sounds from the speaker will have travelled about 22 feet before we return to the 50% factor. We then remain at 50% while the last reverberations die away. We do not have any problem with delay in starting the muting, as we are detecting the threshold before the samples have been sent to the codec input fifo, and are muting them after they have been received by the codec output fifo. Thus we have at least 4ms in which to react.

Even with the muting, the echo is still detectable, but at a more acceptable volume. We still have to keep the latencies as low as possible (50ms round-trip at most) so that the echo seems natural. If it comes say 1/2 second late, then it becomes much more noticeable.

## 5.0 Conclusion

### 5.1 Related Work

There have been many papers describing the setup and control of multimedia streams over a variety of networks. See for example [IEEE90] for a good collection of papers.

Some systems implement real-time voice over synchronous channels [Leung90], while others run receiving video codecs at a higher rate than the transmitting codecs [Casner90] to ensure minimum delay. Systems with adaptive buffering for jitter and clock drift compensation have been proposed, based upon an analysis of recent packet delay times, but these systems often require synchronised clocks, carefully selected parameters or end-to-end cooperation to work successfully. See [Naylor82] for a good analysis of such systems. In contrast, clawback buffers only require the setting of one parameter (related to the rate at which jitter conditions change), operate purely at the destination with no knowledge of the sources, and do not require timestamps.

### 5.2 Future Work

It has been very encouraging that a minimalist approach has continued to operate well while our networking infrastructure and applications have grown. The next implementation (project Medusa) encompasses a wider range of operating environments including Pandora, peripherals attached individually to the network, and application specific software processes running on workstations. While the timing and control of streams will be rather different, particularly if the peripherals are distributed over a large ATM internet, the principles employed in Pandora will still be applicable.

The main difference in Medusa is that the Pandora boards communicating over a network of links and ATM rings have been replaced by Medusa boards communicating over an ATM switch fabric so that we have an *exploded Pandora*. The software running in the ATM switches performs some of the tasks of the Pandora server and network processes, and the same design principles apply. The Pandora boxes themselves have been upgraded to operate over 100Mbit/s ATM links instead of the ATM ring networks, and no retuning was found to be necessary.

The Camera and Audio boards for Medusa operate in a similar fashion to the Pandora boards. The display equipment however is more varied. While we will use Pandora's boxes and special purpose video tablets for display, the majority of video will be displayed on standard workstations equipped with ATM network cards. Here, we find that the Pandora principles are still relevant to the design of device drivers and software toolkits for the real-time streams, and that the overall architecture is very similar in terms of data description and buffering. A major difference is that the streams in a workstation are more independent than in Pandora, being split apart into different chains of processes once they leave the input device driver, and not converging again until they meet the display driver. This makes it much easier to insert special purpose processes such as face trackers into the video paths, but may lead to scheduling problems as control has been delegated to the operating system.

Another approach that we are investigating is to centralise the real-time peripherals in a single workstation. We achieve this by building a number of option cards for the workstation bus. We use the DEC TurboChannel, and have built video capture, compression, framestore and ATM network cards. The cards transfer data by DMA, but are scheduled by the main CPU. This allows us to manipulate uncompressed video with the main CPU, so that user processes can perform operations such as titling, head tracking etc. The same Medusa software infrastructure used in the distributed system controls the streams within the workstation. It operates on similar principles to the Pandora processes, handing around references to the data, and keeping reference counts of the number of processes that are using each segment.

It remains to be seen how difficult it will be to follow these principles as our systems evolve, and how they might have to be extended or relaxed.

## Acknowledgements

Many people at Olivetti Research and the University of Cambridge Computer Laboratory were involved in the Pandora project. The Pandora's Box hardware was designed by Tony King, Oliver Mason, Roy Want, David Clarke, Andrew Jackson and David Milway. The software inside the box was written by Alan Jones, Mark Chopping, Stuart Wray, Gavin Stark and Tim Glauert. The repository was implemented by Ian Wilson. Roger Needham and Maurice Wilkes helped to guide the design.

## References

[Ades86] S. Ades, R. Want and R. Calnan. Protocols for Real Time Voice Communication on a Packet Local Network. IEEE International Conference on Communications 1986, Toronto, June 1986.

[Casner90] S. Casner, K. Seo, W. Edmond and C. Topolcic. N-way Conferencing with Packet Video. In Third International Conference on Packet Video, Morristown, New Jersey, March 1990.

[Hopper88] A. Hopper and R.M. Needham, The Cambridge Fast Ring Networking System, IEEE Trans. Comp., vol 37, no. 10, Oct 1988.

[Hopper90] A. Hopper, Pandora - An Experimental System for Multimedia Applications, ACM Operating Systems Review, vol. 24, no. 2, April 1990

[IEEE90] IEEE Journal on Selected Areas in Communications, special issue on Multimedia Communications, Vol. 8, no. 3, April 1990.

[Inmos88] Occam 2 Reference Manual, Inmos Limited, Prentice Hall International Series in Computer Science, 1988

[Inmos89] The Transputer Databook, Inmos Limited, 1989

[Leung90] W.H. Leung, T.J. Baumgartner, Y.H. Hwang, M.J. Morgan, S.-C Tu. A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks. IEEE Journal on Selected Areas in Communications, special issue on Multimedia Communications, Vol. 8, no. 3, April 1990.

[McAuley90] D.R. McAuley, Protocol Design for High Speed Networks, University of Cambridge Tech. Report no. 186, Jan 1990.

[Nato91] Technical Characteristics of the Navstar GPS, NATO, June 1991.

[Naylor82] W.E. Naylor and L. Kleinrock. Stream Traffic Communication in Packet Switched Networks: Destination Buffering Considerations. IEEE Transaction on Communications, Vol. COM-30, no. 12, December 1982, page 2527.

[Swinehart83] D.C. Swinehart, L.C. Stewart and S.M. Ornstein. Adding Voice to an Office Computer network. In proceedings of GlobeCom 1983, IEEE, November 1983.

[Want88] R. Want. Reliable Management of Voice in a Distributed System. University of Cambridge Computer Laboratory Technical report no. 114, July 1988.

[Wray89] S. Wray, The Interface to Pandora's Box, Tech. Report 89/4, Olivetti Research Limited, 1989.