Presentation Support for Distributed Multimedia Applications

John Bates

University of Cambridge
Computer Laboratory

To my parents: Jean and Derek.

Preface

This report is based on a Ph.D. dissertation submitted by this author in January 1994.

The work documented in this report also features in the following available papers:

The Opera multimedia research group are investigating storage architectures, database paradigms and programming support for emerging multimedia uses. [**Bacon93b**] and [**Moody93**] describe overviews of this work.

Chapter 5 of this dissertation is partly concerned with active objects and their configuration. [**Bates93**] describes an approach to this problem.

A paper based entirely on the work in this dissertation is published as [**Bates94**]. A demonstration video showing experimental applications is also available.

Acknowledgements

I am grateful to all members of the Computer Laboratory who have been of assistance. My Ph.D. supervisor Jean Bacon is wonderful. She has been a great source of encouragement and advice. She has also made many useful suggestions regarding the contents of this report. Ken Moody has good advice for all occasions. Cormac Sreenan, now of Bell Laboratories, has been a great source of help. I enjoyed and benefitted from discussions with Heather Brown, of the University of Kent, during her sabbatical year in Cambridge. Derek McAuley has been helpful with equipment requisition and system advice. Thanks are also due to the following people for reading and commenting on parts of this report: Richard Hayton, Scarlet Schwiderski, Lewis Tiffany, Cormac Sreenan, Robert Sultana, Eoin Hyden and Simon Crosby.

Many thanks to those at Olivetti Research Limited, Cambridge, who have been of assistance. I am indebted to Andy Hopper for the use of a Pandora box in my third year of research. Damian Gilmurray and Andy Jackson have always been ready to help with Pandora system issues.

I would like to thank all my friends at Lancaster University for their continuing encouragement. Andy Colebourne has been particularly helpful with aspects of the X window system.

This work was supported by a grant from the Science and Engineering Research Council.

Trademark Notice

ARM is a trademark of Acorn Computers
Ethernet is a trademark of the Xerox Corporation
NFS is a trademark of Sun Microsystems, Inc
Pandora is ©Olivetti Research Limited
Sesame Street is ©The Childrens' Television Workshop
UNIX is a registered trademark of Novell

Summary

Distributed computing environments can now support digital continuous media (such as audio and video) in addition to still media (such as text and pictures). The work presented in this dissertation is motivated by the desire of application developers to create applications which utilise these *multimedia* environments. Many important application areas are emerging such as Computer-Aided Instruction (CAI) and Computer-Supported Cooperative Working (CSCW).

Building multimedia applications is currently a difficult and time-consuming process. At run-time, an application must manage connections to a range of heterogeneous services to access data. Building applications directly on top of environment-specific features roots them to those features. Continuous media introduces new problems into application management such as control of Quality of Service (QoS) and synchronisation of data items. An application may also be required to analyse, process or display data. Some multimedia applications are event-driven, i.e. they must perform actions in response to asynchronous run-time occurrences. They may also be required to control many workspaces and involve multiple users.

The thesis of this dissertation is based on two principles. Firstly, despite the heterogeneity between and within multimedia environments, that their functionality should be provided in a uniform way to application developers. By masking the control differences with generic abstractions, applications can easily be developed and ported. Secondly, that it is possible to develop such abstractions to support a wide range of multimedia applications. Extensible and configurable facilities can be provided to access, and present multimedia data and to support event-driven applications including cooperative ones.

The approach taken in this work is to provide a *presentation support platform*. To application developers this platform offers an authoring interface based on data modelling and specification using a script language. Using these facilities, the parts of an application involving interactive presentation of multimedia can be specified. Services have been built to support the run-time realisation of authored presentations on top of environments. Experiments show that a wide range of applications can be supported.

Terminology

Active Context A script activated by an application is applied to a database context yielding an active context.

Active Object An object with data, methods and autonomous threads of control. It can send and receive messages asynchronously.

Application A user program.

Application Developer A user of authoring facilities.

Attribute A named user-defined characteristic of a data object. Each has an associated value.

Authoring The process of specifying the data and activity involved in a multimedia presentation.

CAI Computer-Aided Instruction.

CAL Computer-Aided Learning.

Configuration A combination of interconnected multimedia presentation objects.

Context A group of related data objects.

Context Group A collection of active contexts to which invocations can be sent.

Continuous Media Dynamic temporal media such as audio and video.

CSCW Computer-Supported Cooperative Working.

Data Item Raw data of a particular medium and format, located on a source device.

Data Modelling The creation, tailoring and relating of data objects.

Data Object An encapsulation of a multimedia data item. It can be tailored with events and attributes.

DCE Distributed Computing Environment- composed of processors running operating systems, connected by networks for communication and data sharing. Multimedia DCEs are capable of supporting continuous media (such as audio and video) as well as still media (such as text and pictures).

Environment See DCE.

Event An occurrence associated with multimedia data or the environment.

Event Expression The condition part of an event rule- one, or a temporally ordered combination, of events. If it is detected, it causes the event rule to fire.

Event Rule Named rules involving conditions and actions. Actions are invoked if conditions are evaluated to true.

Factory An object which manages the creation of one or more classes of active presentation object.

Factory Builder An object which manages the creation of factories.

Factory Site Location in which factory builders, factories and created active objects can exist. Sites must have processing resources for their objects' threads of control.

Firing Event rules fire when their event expressions are satisfied.

Format The encoding of a data item.

Group Invocation Invoking a method on a set of objects.

IMP Interactive Multimedia Presentation.

Interaction User involvement with an application.

Interface A set of methods, used to control an object at run-time.

Interface Reference Through possession of an object's interface reference a client can invoke any of the methods in that object's interface.

Lip-Sync Slang phrase specifying a correlation between the actions in a video clip and the sound in a related audio clip.

Marked Event A named property of a data object, indicating an associated run-time occurrence of interest.

Multimedia A mixture of media types such as text, pictures, video and audio.

Notification The act of informing an interested party that an event has occurred

Presentation The parts of a multimedia application concerned with concurrent device access, processing, analysis, synchronisation, display and event handling for multimedia data items.

Presentation Object An abstraction which manipulates data in some way such as retrieving it from a device, synchronising it, displaying it, processing it or analysing it.

Presentation Platform Provides uniform abstractions for authoring multimedia presentations. Also presentation facilities to realise these at run-time.

QoS Quality of Service- an abstraction of the amount of DCE resources (e.g. processor, network or service bandwidths) used in a presentation. It is adjustable based on the availability of resources.

Script A program written in a script language is used to specify the interactive presentation of data objects.

Set A group of objects which can be manipulated as a whole.

Still Media Static media such as text and pictures.

Stream An abstraction representing the flow of continuous media from a *source* to a *sink*.

Synchronisation Intra-stream synchronisation involves jitter removal. Inter-stream synchronisation involves a temporal correlation between streams, such as audio and video which must be lip-sync'ed.

Tool A program to manipulate presentation platform facilities.

Workspace A parent window into which display objects can be laid out.

Table of Contents

Chapter 1

Introduction

Distributed Computing Environments (DCEs) are composed of autonomous processor nodes connected by a network to allow communication and data sharing. Processors run operating systems to manage the scheduling of processes and communication between machines. The user community of DCEs usually share system resources such as storage services. Single user workstations represent the current trend in processor nodes. Workstations have a bit-mapped screen which allows them to run a windowing system, and peripherals such as a keyboard and mouse. Until recently the components of DCEs were only capable of handling still media such as text and bit-mapped images. With recent increases in processor and network speeds they are now capable of handling continuous media such as digital audio and video.

The term *multimedia* has emerged to describe a graphical workstation environment in which multiple medium types can be manipulated and displayed. Figure 1.1 illustrates such an environment. Workstations have been augmented with speakers, cameras and microphones for the input and output of audio and video data. Storage services and networks have been extended to support multi-service data. For example a user can send the output from a camera to a video storage service. Anyone with valid access can play that video in a window on their workstation screen. Display of multimedia items is achieved by transporting the data from its source, into the user's workstation and displaying it in a window. A user sitting at his/her workstation can have windows showing video from cameras in other offices, another containing current television pictures with sound coming through the speaker and others containing text and pictures.

Such an environment creates the possibility of an exciting new range of applications. There is an increasing user demand for applications such as *hypermedia documents* and *conferencing systems*. It is becoming apparent that multimedia is going to have major influences in the areas of Computer-Aided Instruction (CAI) and Computer-Supported Cooperative Working (CSCW).

1.1 The Research Problem

Due to the complexity added to environments by support for continuous media, development of distributed multimedia applications is currently a difficult and ad hoc process.

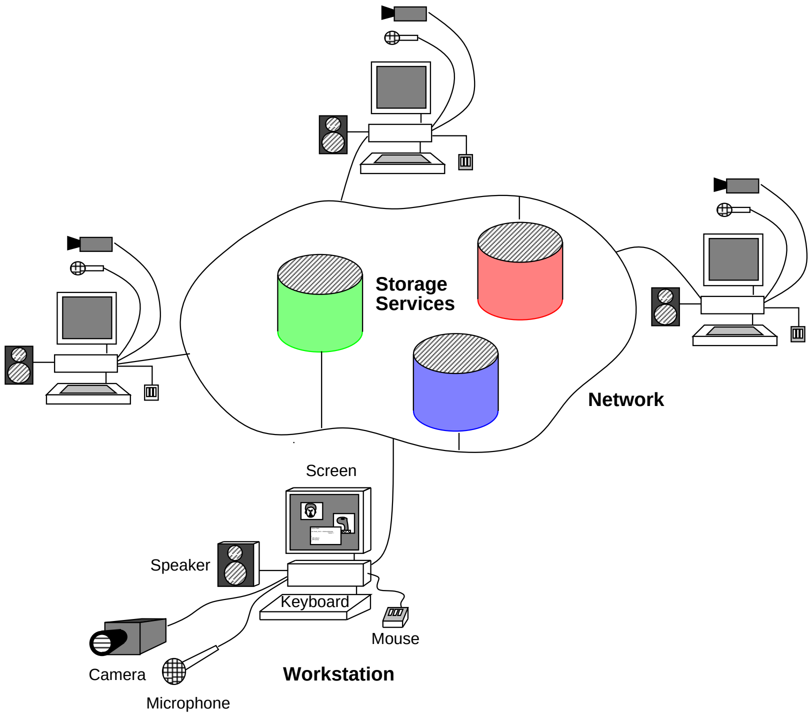


Figure 1.1: User view of a multimedia environment

1.1.1 Characteristics of Multimedia Environments

Multimedia environments are still of an experimental nature. Methods for handling continuous media and emerging concepts such as Quality of Service (specifics of multimedia environments are described later in section 2.2.1) are constantly evolving. Therefore the characteristics and interfaces of components are constantly being re-evaluated and updated. As services evolve it is often desirable to continue to support older services since applications may rely on them. This leads to an environment with many heterogeneous services with different interfaces and different levels of functionality.

1.1.2 Application Development Techniques

Using traditional development techniques, a multimedia application includes library interfaces for the devices and services it is required to connect to and control. Due to the experimental nature of multimedia environments, applications to illustrate their facilities are often developed on a *per application* basis.

Applications must manage a complicated range of run-time tasks, which must be specified by the application developer. They may be required to retrieve and display many different medium types, such as text, pictures, video and audio, each of which may have a different format (encoding). Presentation functions such as processing, synchronisation or analysis of data may be required before display.

Additional complexity may be introduced when using event handling facilities. Some multimedia applications generate asynchronous events which can be used as the trigger to initiate activities. Some programming toolkits allow the specification of handling routines, to be called if a certain event occurs. The run-time systems of such toolkits often take over the application threads of control. It is thus difficult to use two such systems within the same application. Examples of such systems are toolkits for the X window system [**Scheifler86**] and the ANSA testbench [**ANSA90**].

In summary, application structure tends to reflect management of the underlying system rather than the function of the application.

1.1.3 The Inadequacies of Application Development Techniques

Using the application development techniques described presents problems to long and short term software development, for the following reasons:

- *The complexity involved in application development*
  It may be necessary to use the facilities of many heterogeneous components and specify many complicated run-time tasks.
- *The lack of application portability*
  Directly referencing a specific interface ties an application to that device or service. This makes applications difficult to port between different environments.
- *The lack of application maintainability*
  Within the same environment, if devices or services are changed or their characteristics updated, applications may require time consuming updates. The unrepresentative nature of application structure increases maintenance difficulties as the code may be difficult to follow.
- *The lack of application component reusability*
  Development on a per application basis leads to a lack of reusable abstractions for long term support.

1.2 Research Aims

The research outlined in this dissertation investigates authoring and run-time presentation support for multimedia applications in a distributed context. This work is motivated by the problems highlighted above. It is based on the belief that a set of generic requirements can be obtained from current and emerging multimedia applications. Using these requirements it is possible to formulate an approach to authoring which can be used to build a wide range of multimedia applications. Application construction must be relatively easy and completed applications portable and maintainable. The term *interactive presentation* is used to describe the part of a multimedia application concerned with connection to multimedia devices, multimedia data presentation and event handling.

Figure 1.2 illustrates the approach investigated in providing support for interactive presentation. A *presentation support platform* has been developed. *Application developers* are clients

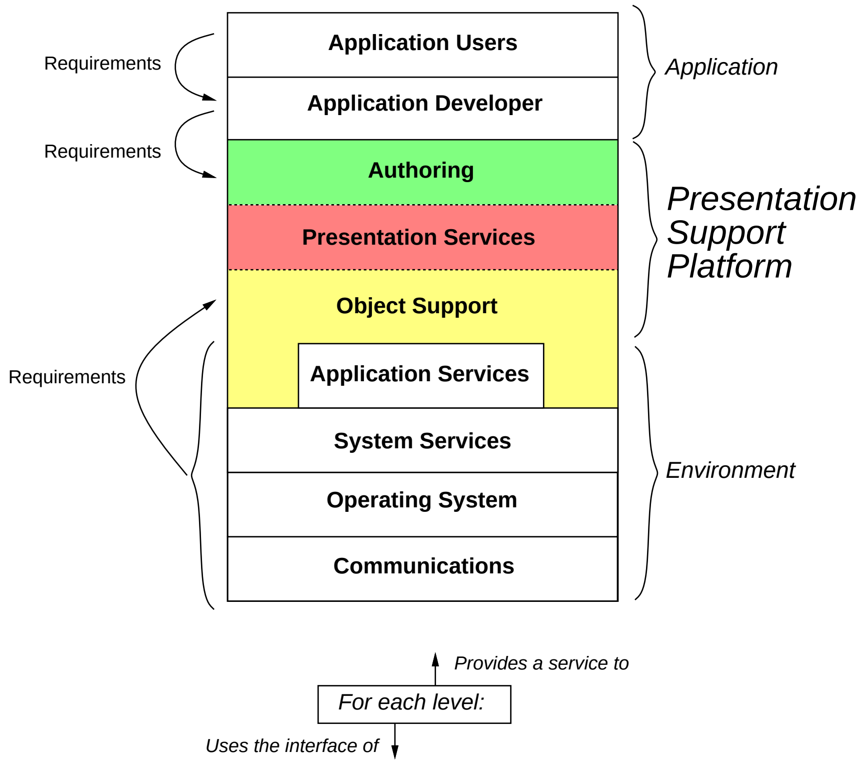


Figure 1.2: The role of presentation support

of this platform which offers them interactive presentation authoring facilities. The platform uses presentation services and object support to coordinate multimedia environments during presentations. The diagram illustrates how the requirements of application users drive the application developers' requirements of an authoring system. The design of object support and presentation services is shaped by the control requirements of the environment.

1.3 Dissertation Overview

The structure of the remainder of this dissertation is as follows:

Chapter 2 establishes the features that a presentation platform must provide to the application developer. An assessment of existing and emerging multimedia applications is first provided. Then the facilities of multimedia environments are outlined. From this, the generic run-time responsibilities of multimedia applications within an environment are assessed.

Chapter 3 describes related work in the area of multimedia application support. It discusses the benefits and shortcomings of existing approaches.

Chapters 4, 5 and 6 illustrate design requirements of authoring and presentation facilities. Designs are illustrated with reference to the IMP (Interactive Multimedia Presentation) prototype platform developed within the Cambridge multimedia environment. Chapter 4 illustrates an authoring approach which fulfils the requirements highlighted in chapter 2 and takes account of the shortcomings of approaches in chapter 3. Chapters 5 and 6 illustrate the design of services to implement interactive presentation. Chapter 5 describes how objects representing multimedia data and presentation functions on that data are realised. Chapter 6 describes how high level services use these objects to implement interactive presentations.

Chapter 7 outlines the configuration of the prototype IMP platform and experiments carried out using it. Applications developed come from application classes illustrated in chapter 2 including hypermedia and conferencing systems.

Chapter 8 shows how the research goals have been met by the work presented and suggests further developments of the research.

Chapter 2

Platform Requirements

In this chapter the requirements of a presentation support system are assessed. To achieve this first requires an analysis of the interactive presentation requirements of existing and emerging multimedia applications. A study of the environments in which they must exist allows a formulation of the management responsibilities of an application. Conclusions are drawn as to the correct approach in providing presentation support.

2.1 Multimedia Applications

A review of the main types of multimedia application will be used to assess their support requirements. Uses range from applications which simply use multimedia to enhance their user interfaces to applications which are built around interactive multimedia processing, analysis and display. The distinction must be made between

- *Types of application*
  Meaning a sub-division of the ways in which applications are structured.
- *Uses of individual application types*
  Meaning the real world areas in which particular types of application are emerging as useful.

A good review of the uses of multimedia is found in [**Blair92**]. This summarises multimedia application uses as

- *Office automation*
- *Service industry applications* - such as in education, financial services and health
- *Retail applications* - such as in publishing, travel and property
- *Domestic applications*
- *Science and engineering*
- *Cultural activities*

The study presented here concentrates on the types of applications which have been developed since it is from this that their support requirements can be analysed. References to how these types are being used in such application areas will be included.

This overview has been compiled using existing multimedia applications and the perceived nature of emerging applications. Multimedia applications developed as part of the Pandora project[1] [**Hopper90**] provided a first-hand view of application characteristics and usage patterns.

The environments being used to develop these applications range from single user systems to distributed multimedia environments. This study is targeted at the latter but subsumes the former.

A general distinction between types of application is shown below. Each category described subsumes the previous one.

- *Presentational*
  Applications involving the editing, processing, analysis or display of multiple media items. Media items have a medium type which may be, for example, audio, video, text or picture. This type of application is described in section 2.1.1.
- *Compositional*
  Applications which create relationships between multimedia items involved in a presentation. Hypermedia documents involve the composition of multimedia items and associated timing, synchronisation and interactive selection information. They are described in section 2.1.2.
- *Collaborative*
  Applications involving multiple contexts for the presentation of multimedia items and relationships between the users of these contexts. These applications can be divided into those exhibiting asynchronous interaction, such as multimedia mail systems (see section 2.1.3), and synchronous interactions, such as multimedia conferencing (see section 2.1.4). Facilities for group editing such as shared hypermedia systems have also been investigated (see section 2.1.5). Emerging multimedia applications (see section 2.1.6) will involve many of these developing technologies.

Compositional and collaborative applications still involve the problems of multimedia presentation, and collaborative applications may still involve composition.

2.1.1 Presentational Applications

The functions which may be involved in a presentational application are illustrated in figure 2.1.

Developments in digital continuous media have enabled many applications to use multimedia as a *front-end*, enhancing their user interface. This can be achieved by retrieving and presenting video and audio information to augment the traditional media of text and pictures.

[1] A joint project between Cambridge University Computer Laboratory and Olivetti Research Limited, Cambridge

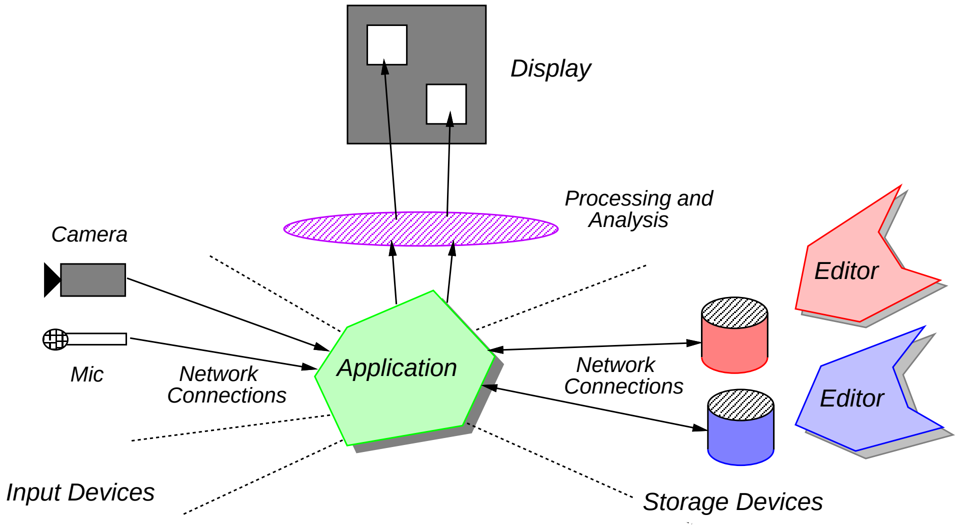


Figure 2.1: Presentational application concerns

Live information services display digitised information from live sources such as cameras, microphones, Compact Discs (CD) and television. As part of the Pandora project an application called *XVmedia* has been developed. This allows a user to connect to and interactively select from live information services including television, radio and a CD jukebox. Video and audio information are digitised and presented to a user in workstation windows and via a workstation speaker.

Another Pandora application called *XVlook* has been developed which allows a user to select a view through any network camera. Such remote monitoring using cameras has applications for security surveillance and, when used in conjunction with audio, represents the basis for conferencing systems (see below).

Many direct-manipulation editing tools have been developed for text and bit-mapped pictures. Editing of continuous media data is complicated by its temporal nature. Most editors developed for video and audio, such as in the Pandora project, have involved selection of *clips* which are arbitrary sequences. These can be joined together using cutting and pasting operations. Such tools can be used to prepare media items for use in an application.

Some applications involve dynamic processing of multimedia information. 'Desktop' editing of such data enables elaborate presentational effects to be achieved. A description of some techniques can be found in [**Cook90**]. Application areas range from television studio processing to domestic usage.

Some applications require dynamic analysis of multimedia data. Information patterns can be detected without the need for human intervention such as in image and speech analysis. This recognition has a wide variety of uses such as speech-driven computers for people with bad mobility and visual pattern-matching for industry production lines, scientific purposes or automated targeting for military applications. As part of the Pegasus project [**Leslie92**] applications are being developed which can detect the positions of people using live video and stereo audio information.

2.1.2 Hypermedia Documents

The notion of a hypermedia document has evolved from hypertext (for a good survey see [**Conklin87**]). A document is a collection of information elements. A hypertext document provides electronic representation of a textual document and allows links between information elements to be added. Links can be navigated by selecting marked items of text. This usually provides access to the referenced data. It is possible to create complex graph structures of links between text elements. As an simple illustration, a reader may be scanning a page of text about universities and Cambridge may be highlighted as a link. Selecting it could display a page of text on Cambridge.

Hypermedia integrates other media into this model. A good description of how hypermedia should extend the hypertext paradigm is given in [**Fujikawa91**]. As well as simply displaying text-based information, hypermedia allows other media to be presented such as video, audio and still pictures. These media items can be selected using links in a similar way to hypertext. When a link is navigated, by selecting part of a media item, other media items can be presented. Another concept introduced by the hypermedia model is the *timeline*. Events such as the start and end of the presentation of a media item can be used to start the presentation of other items.

Hypermedia document applications represent very good learning aids. The terms Computer-Aided Instruction (CAI) and Computer-Aided Learning (CAL) have been used to describe tools which present educational information through direct manipulation. They are also a natural extension of electronic publishing, providing electronic documents incorporating audio and video with text and pictures. These applications have also been used to represent information systems such as on-line catalogues. Some applications, such as those developed in the Muse system [**Hodges89**, **Mackay89**], are *scenario-based*. In such applications a user can move between and interact with a series of 'scenes'.

International standardisation has recognised the importance of hypermedia documents. The HyTime [**ISO/IEC93b**] and MHEG [**ISO/IEC93a**] standards are designed for hypermedia information interchange between applications.

2.1.3 Multimedia Mail Systems

Electronic mail systems are now widespread in educational establishments and businesses. Text messages can be composed and transmitted to other named users of a network. Mail systems which allow transmission of composite multimedia messages have also been investigated. Both the sending and receiving mail applications must have knowledge of the message format.

The DARPA experimental multimedia mail system [**Reynolds85**, **Postel88**] allows messages composed of text, bit-mapped pictures and audio to be transmitted to users over an internetwork. The *XVmail* system developed as part of the Pandora project allows a user to augment standard text-based mail with video and audio. A textual message is sent along with pointers to the related video and audio which is stored on a central service.

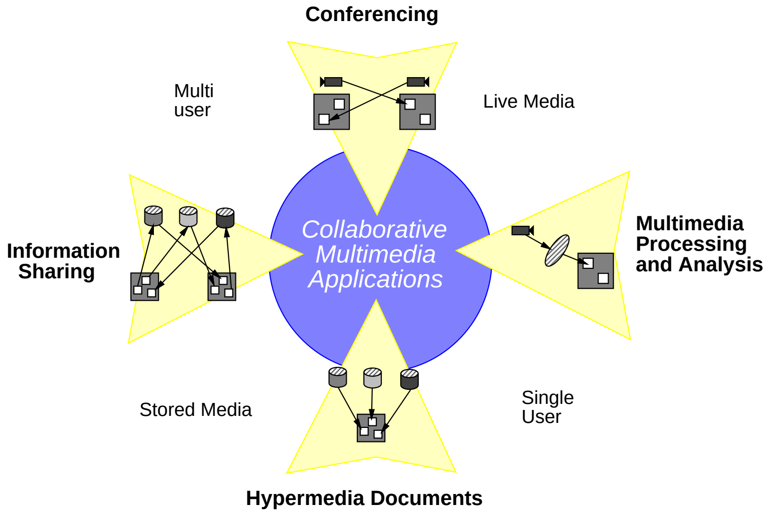


Figure 2.2: Emerging trends in multimedia applications

2.1.4 Conferencing Applications

This class of application involves interaction between two or more users. As with the information presentation process, multimedia has enriched the possibilities for inter-personal communication.

Text-based conferencing systems have been developed, such as the UNIX *talk* program in which two users each have a text window to type into and another to see the output of the other user.

Shared design tools have been developed to allow distributed users to collaborate in visualising ideas. Using a shared drawing tool [**Greenberg**, **Tang**] each user is presented with a drawing area and a selection of drawing tools. Any drawing strokes made are propagated to every user's display.

There has been much recent interest in conferencing through the use of audio and video. Example systems are *XVphone* developed in the Pandora project, *MediaSpace* [**Bly93**] developed by Xerox PARC and *Cruiser* [**Fish93**] developed by Bellcore. Such systems allow two or more users to carry out a conversation whilst seeing and hearing the other conferencees.

Conferencing applications provide alternatives to the phone call for domestic and business use. They provide new methods for formal or informal meetings between geographically separated colleagues.

2.1.5 Shared Editing

Models for the shared editing of documents allow geographically separated colleagues to collaborate in the authoring process. For example, in shared hypermedia systems, different users can insert nodes and links [**Meyrowitz86**, **Haake92**]. Facilities must be provided to deal explicitly with the conflicts in this process of interaction.

2.1.6 Emerging Cooperative Applications

The term Computer-Supported Cooperative Working (CSCW) has been used to describe applications involving multiple users cooperating in both information sharing, processing and communication. They may thus exhibit properties of any of the above applications. Figure 2.2 indicates how types of application are merging to create a desire for collaborative interactive multimedia applications. These applications will provide *virtual workspaces* in which users can move around, converse and share ideas. A good overview of the technical challenges posed by CSCW to multimedia environments is given in [**Blair93**].

2.2 Distributed Multimedia Environments

This section describes how features of a multimedia environment are realised as system components and how such components are controlled. This highlights the responsibilities of an application which uses such an environment.

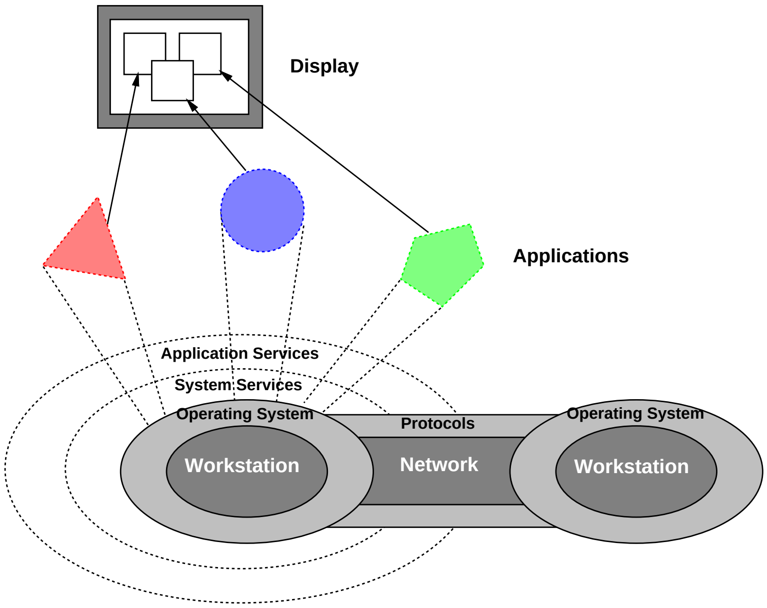


Figure 2.3: Multimedia environment components

Distributed environments have been altered by the integration of continuous media. The reasons behind this are first described followed by a study of system components which have been affected. Figure 2.3 illustrates the main layers of such an environment. It shows how applications are built on top of an operating system. They can access and process media items using system and application services. Data transport is realised using network transport protocols.

The different approaches taken in the construction of these components are too many and varied to be described here. Despite the different implementation approaches taken, all environments aim to provide the same general component functionality. This survey will use illustrative examples from the Cambridge multimedia environment. This is a typical environment and introduces the setting in which the research described in this dissertation was carried out. It should be noted that some components are still in a research stage and are not stably available for use.

2.2.1 The Nature of Continuous Media

Support for continuous media has changed the nature of distributed environments. Continuous media items are made up of a number of sub-elements which must be presented continuously over a period of time. Figure 2.4 illustrates how continuous media travel in *streams* from *source* to *sink*.

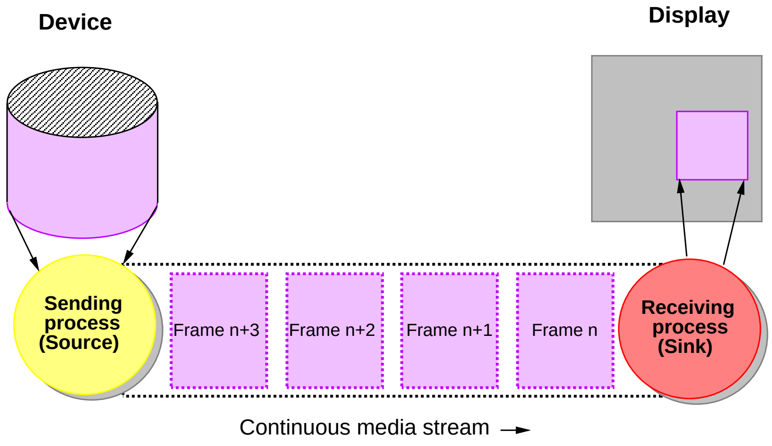


Figure 2.4: Continuous media

Data formats describe the way in which data is encoded. All multimedia data must be coded in a certain way so that in storing, retrieving, transmitting and receiving all parties know how to encode, decode and process the data. International standardisation has led to widely recognised formats for information interchange between a wide range of applications. Formats for video and audio encoding and compression have been developed such as JPEG [**Wallace91**] and MPEG [**Gall91**]. However, other encodings are still used, such as the Pandora video and audio formats [**Wray89**].

Continuous media creates new problems because it has the following characteristics:

- *Continuous nature*
  Unlike traditional data traffic, instances of continuous media traffic may utilise resources for a possibly indeterminate length of time.

- *Time deadlines*
  As described, continuous media streams are made up of a series of elements representing data, such as frames of video. Streams have certain presentation requirements such as a video stream whose elements must be presented at the rate of 25 frames every second.

These characteristics can cause congestion in both network transmission and individual processors. Faster hardware can improve this situation but there is no guarantee that hardware will ever be adequate to support the desirable performance levels. An environment must thus embody the principle of Quality of Service (QoS) [**Nicolaou90**, **Campbell93**]. QoS describes the ability to adjust the quality of data presentation in response to resource availability. Environment components embody policies for the allocation of their resources. For instance, if a processor or network cannot support the presentation of high quality video at 25 frames per second, then an application must accept a lower rate of presentation.

2.2.2 Hardware Support

The desire to experiment with multimedia before processors were fast enough led to the development of special hardware to process continuous media.

The Pandora box [**Hopper90**] is an example of such a device, developed by Olivetti Research Limited and Cambridge University Computer Laboratory. This device has a connection to the Cambridge Fast Ring (CFR) [**Hopper86**], a high-speed ATM network. The box also has camera, microphone and speaker attachments. It displays video by overlaying video windows directly onto the video signal from the workstation before it is displayed on a monitor. Pandora boxes can display video from the local camera, audio from the local microphone or make network connections to other boxes and display information from their devices. A prototype Pandora video repository has been implemented and is described later.

It should be noted that general purpose processors are now becoming fast enough to display continuous media in real-time, especially with the assistance of hardware compression boards.

2.2.3 Communications Support

Asynchronous Transfer Mode (ATM) networks are most suitable for carrying *multi-service* traffic due to their small fixed packets (cells), fairness of cell transmission policy and flexibility of cell path selection. Protocols have been developed to capitalise on these ATM features. Network services such as QoS managers can be contacted to negotiate an initial level of network bandwidth for a session. In some approaches the network may require this to be renegotiated; in others it is fixed for a session.

In the Cambridge environment, the Multi-Service Network Architecture (MSNA) [**McAuley89**] provides a set of protocols for the transport of multi-service traffic. Research is continuing into providing QoS negotiation services [**Crosby93**].

2.2.4 Operating Systems Support

It is desirable to provide operating systems for multimedia environments to make use of ATM network technology and increasingly fast processors. Using such systems, it must be possible to run programs which process continuous media interchangeably with traditional programs. Facilities for QoS must be provided to allocate processing resources based on application characteristics. Applications are expected to handle events which claim resources back from them. An example of such a research project is the Pegasus system [**Leslie92**, **Hyden94**].

2.2.5 System Services

The design of system services such as the management of storage has developed to incorporate continuous media. This has involved techniques such as improving efficiency in disc access and QoS policies to manage available bandwidth. Despite the availability of these new services, existing ones must often be maintained due to the amount of data kept on them and the number of applications which rely on them. This has created a heterogeneous system environment.

Such a situation exists in the Cambridge University Computer Laboratory. Sun's Network filing system (NFS) is used with the UNIX operating system as the standard program development environment. Files are arranged into hierarchical directories and accessed with a pathname. Many applications are based on this system. Several prototype storage systems have been developed. A Pandora video repository[2] [**Wilson88**] is able to store Pandora format video and audio. These can be accessed using a video repository pathname. The *Multi-Service Storage Architecture* (MSSA) [**Lo93**] is a scalable storage model for traditional and continuous media data. At the lowest level it provides a byte-stream storage abstraction. Above this, *custodes* may be provided to manage data of different types. Data objects are accessed via storage service identifiers (SSIDs). The architecture embodies the principle of QoS on a *session* basis. An application obtains a *ticket* which gives rights of access to resources throughout a session.

2.2.6 Application Services

Services which provide facilities to assist applications in managing continuous media presentation functions have been investigated. One such presentation function is synchronisation. Using packet networks to transport continuous media information leads to the problems of jitter and skew. Jitter is characterised by temporal inconsistencies in the presentation of continuous media elements such as video frames and audio samples. Skew is a misalignment in the presentation of related streams such as a video with an audio soundtrack. Synchronisation services have been developed which remove jitter and skew from streams.

An example of such a service [**Sreenan93**] provides *stream* and *group agents*. *Stream agents* use buffering to remove jitter in a particular stream, given details of stream format. *Group agents* can synchronise the processing of stream agents using their externally available timing information thus removing skew. QoS parameters such as application tolerance to jitter and skew can be specified.

2.3 Application Responsibilities

This section assesses the generic responsibilities of a multimedia application with reference to interactive presentation. The major classes of multimedia application have been outlined along with environments in which they must execute. The assessment in this section is performed with the belief that despite functional differences between applications, their

[2] Located at Olivetti Research Limited, Cambridge. Accessed from Computer Lab via CFR.

interactive presentation requirements are generically similar. Through this analysis a set of requirements for presentation support can be formulated.

The responsibilities of an application within a multimedia environment can be summarised as

- Managing connections to heterogeneous devices
- Managing information presentation
- Response to events

Issues relating to these responsibilities are discussed below.

2.3.1 Managing Connections to Heterogeneous Devices

A major application development problem is that of heterogeneity. Before multimedia, application development environments such as UNIX had the advantage of homogeneous component interfaces for the manipulation of the operating system and storage abstractions. Multimedia environments are characterised by heterogeneous devices with varying interfaces and characteristics. Continuous media support has increased application complexity through the need to manage concurrent, network and processor intensive data manipulation and the associated issue of QoS.

An application must make network connections to and control the flow of information from multimedia devices such as cameras, microphones and storage services. If such devices embody the concept of QoS, an application will first have to negotiate a level of QoS for the session. Dependent on policy, it may be required to renegotiate this at a later point. The application is also responsible for making sure that functionality is maintained without exceeding this QoS level. To achieve this it must manage policies for controlling its usage of resources. A possible scenario is that an application has five video and five audio connections open, and it does not have enough bandwidth to open a new one. The application must have knowledge of user priorities in this situation. It could be configured to degrade the resolution of video connections to 50% and cut the frame rate to 10 frames per second. If this does not free enough bandwidth, then the required policy may be that the audio connections be cut.

Due to the evolving nature of environments, many of the early applications developed have been targeted towards specific devices. The aim was to prototype rapidly an illustration of novel device features. This led to applications rooted in particular environment features which have little potential for reuse.

2.3.2 Managing Information Presentation

The application is responsible for managing the analysis, processing and display of multimedia data. Display involves placement of relevant window abstractions on the workstation screen and putting data into them. Processing or analysis of the data before it is displayed may be required, for example, to perform image recognition algorithms on pictures or to improve the quality of audio. The application must also manage intra and inter stream synchronisation of multimedia data.

It is possible to construct application services to control functions such as synchronisation, image processing and image recognition. It is still up to an application to control the flow of streams through these services. Even components of the same medium type may not always be processed using the same algorithm. To process, synchronise or display the data, the data format must be known.

2.3.3 Response to Events

It cannot be assumed that application flow of control will be sequential. An application must be ready to handle specified asynchronous occurrences (events). Three main generators of events are

- *User interaction*
  In applications such as hypermedia documents the application must respond to interaction from the user such as clicking on words of text.
- *Points of presentation*
  Applications can be required to respond to signals such as a certain time value in the presentation being reached or a certain frame of video being presented.
- *System signals*
  Environmental occurrences can generate events. For example, an application can be notified of a system request for it to revoke or limit its use of certain resources.

Such multimedia applications are termed *event-driven* in that it may be desirable to initiate actions within an application if specific events occur. Within cooperative applications, event occurrences in one user's context can have effects on other users' contexts.

Due to the many different forms of event and the lack of standard event handling mechanisms in existing programming languages, the handling of events can become an ad hoc process.

2.4 Analysis and Conclusions

The major classes of existing and emerging multimedia applications and the nature of multimedia environments have been outlined. From this analysis the following conclusions are drawn:

- Despite the fact that applications appear to fulfil many different purposes, their responsibilities within multimedia environments can be generalised, as described in section 2.3. If all of these responsibilities are duplicated on a per-application basis then work is being replicated unnecessarily. Also applications are being tailored to specific environments which means they are not portable between environments or upgradeable as environments develop.

- Despite environments being diverse in characteristic their aim is the same. Environments aim to support multimedia applications which control devices associated with live and stored multimedia data, its transport to the workstation, its analysis, processing, synchronisation and presentation. Thus environments are similar in function although different in interface. Taking storage services as an illustration of this point, there is a large array of storage architectures available for multimedia environments. Despite differences of interface they attempt to provide similar services of storage, retrieval and manipulation of multimedia objects.

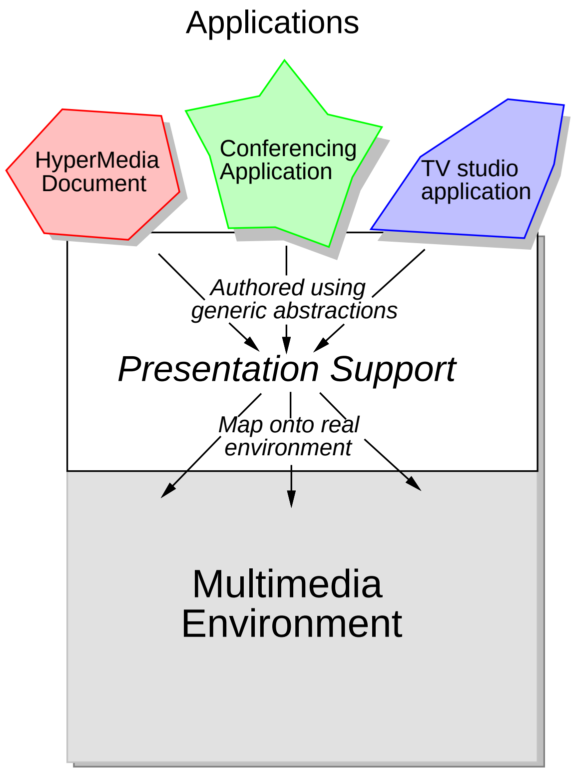


Figure 2.5: Multimedia application support

The work presented in this dissertation addresses these points by means of an application authoring approach based on generic abstractions (chapter 4). A wide range of applications can be supported using these abstractions, which can be tailored to suit individual circumstances. Environment complexities are masked from the user and device access, presentation functions and event handling can be specified generically. Figure 2.5 illustrates how different applications can be developed on top of a set of generic abstractions. A set of services are used to realise authored applications at run-time (chapters 5 and 6). These manage the difficulties of controlling a particular environment.

Chapter 3

Related Work

The rationale behind presentation support for multimedia applications has been outlined. Without it applications must be crafted directly onto environments, which root them to that environment. Features which all multimedia applications must manage can be generalised and provided as presentation support.

In this section work which has attempted to fulfil these criteria is outlined. This analysis has been structured into a gradual evolution of concepts. There are two main aspects to any presentation support work- authoring and presentation. An authoring system provides a model to express how interactive presentation should occur. A presentation system realises interactive presentations at run-time. The work discussed here addresses either or both of these problems. The benefits and limitations of each approach are pointed out and evaluated with respect to the approach presented in this dissertation.

3.1 Project Athena at MIT

Athena Muse [**Hodges89**] is aimed at the construction of educational software. Tools shaped by particular application classes [**Mackay89**, **Hodges91**] have been built on top of Muse.

3.1.1 Muse Construction Set

Muse provides a multimedia construction set for building learning environments. The following four techniques are provided for application development:

- *Directed graphs*
  Media items can be grouped into information packages. When a package is activated its display elements are presented. Directed graphs can be used to link packages together. *Hypermedia links* can be set up, which are fired by sending activation signals from one node to another. *State transition networks* can be used to represent states in an application, e.g. in a *branching movie* where users can interact with a series of scenes.

- *Multi-dimensional spatial frameworks*
  Movement in multiple dimensions can be represented. The simplest example is a *timeline* which provides facilities to temporally synchronise different media, e.g. sub-titles synchronised to a video. Spatial control is also possible such as positioning, zooming

and panning. Multiple dimensions can be used for simulations, e.g. simulating the x,y and z axes as well as time and angle of view.

- *Declarative constraints*
  Declarative constraints specify functions which automatically change one abstraction if another is affected. An example is using a scroll bar to move *temporally* through a presentation or updating the scrollbar as time proceeds normally.

- *Procedural language*
  A language known as *Event script* can be used to attach special behaviours to display elements. It can interface to the X window system to use X events to control objects.

3.1.2 Tools and Applications

Example applications have been developed using Muse. The *Navigation project* made use of the multi-dimensional space model. It allowed users to move in many directions around pre-defined scenarios. In response to user interaction information was drawn from a video disk.

The following tools to make certain classes of application easier to develop have also been created:

- *Interactive documentaries*
  An interactive film viewing and editing tool has been developed. This can combine video segments with text annotations for documentary development.

- *User interface research*
  An Experimental Video Annotator (EVA) was written in Athena Muse. This allows the live annotation of video with textual messages allowing points of interest to be noted for later analysis.

- *Multimedia communication*
  Pygmalion, a multimedia message system, has been built. Users can send messages which are all stored at a central post office. A pointer to the message is sent to the intended recipient. A *neuroanatomy research database*, which allows manipulation of views of the human brain, has also been developed.

- *Plastic editors*
  MuseBuilder [**Hodges91**] is a combination of editors to assist application creation. It allows the appearance and behaviour of applications to be graphically manipulated. Editors are themselves built out of Muse tools.

3.1.3 Evaluation

The Muse project and environment was one of the first undertakings to recognise that multimedia applications required generalised support. The model is only sufficient to represent the educational applications for which it was intended. It includes features which are useful in the development of compositional applications without a developed architecture for specifying presentational concerns.

3.2 Microelectronics and Computer Technology Corporation

A multimedia database, ORION, has been developed [**Woelk86**] which can store video, audio, text and pictures. Relationships can be represented between objects for multimedia document applications. The novel aspect of this work [**Woelk87**] was in recognising that multimedia databases must provide an application developer with additional support for capturing, presenting and storing such objects.

3.2.1 ORION: An Object-Oriented Multimedia Database

Both objects and classes are stored in the database and can be related to represent *multimedia documents*. An example is a memo class which contains different fields - text of sender, recipient, message body, related audio etc. Objects of this composite class can be instantiated as actual memos. The types of relationship available are listed below.

- HAS_ATTRIBUTES
  Used to specify attributes of classes, e.g. `fontsize = 14`.
- IS_TYPE_OF
  Used to create a sub-class of an existing class. These inherit characteristics such as font size for a text class.
- HAS_PARTS
  Used to reference parts of a composite object. This represents an instantiation of the CAN_HAVE_PARTS relationship which is used to define optional parts of a composite class.
- IS_INSTANCE_OF
  Used to reference instances of composite object templates.
- HAS_METHODS
  Used to start independent programs. For example a `SEND` method could be used with the memo object to send it via electronic mail.

3.2.2 Multimedia Information Management

This work is based on the belief that future databases must provide the capability for applications to capture, store and present multimedia information. The design objectives for this system are extensibility and flexibility. New types of devices and protocols can be added, e.g. a new type of colour display, camera or audio digitiser. Also it should be possible to specialise devices, e.g. a window is a specialisation of a display. Control over capture and presentation such as being able to pause, continue, fast or slow wind during playback must be provided.

Applications in the database environment can use objects of the following types:

- *Presentation device objects*
  An instance of a presentation device object represents a physical device to present on, where on that device to present and what part of an object to present. The class hierarchy includes spatial devices, such as for image and text, and linear devices for audio.

- *Capture device objects*
  These objects can be used to control capture devices and specify spatial and linear attributes.

- *Captured objects, storage device objects and disk stream objects*
  Every multimedia object stored in ORION is an instance of a class captured object or one of its sub-classes. Storage device instances manage the reading and writing of data to and from disk. Data is transferred to and from storage via disk-stream instances. These are created when they are needed.

Objects communicate by *message passing.* For example, to present or capture information, a presentation or capture objects sends a message to a particular storage object to open for read or write. A disk stream object is created to manage data transfer. The disk stream object controls the flow of data to and from the capture, presentation and storage devices. Captured image objects keep a reference to their related storage objects.

3.2.3 Evaluation

Many data models for representation and composition of multimedia objects have been proposed [**Bertino88**, **Christodoulakis86a**, **Christodoulakis86b**]. This work was the first to recognise that multimedia databases could not merely be concerned with passive data entities. Run-time objects are created to manage capture, presentation and device access for continuous media data. Although this model only assists the development of presentational applications (section 2.1.1), it should be noted as an important milestone.

3.3 Osaka University

Harmony [**Fujikawa91**] provides an extension to the hypertext paradigm for multimedia presentations.

3.3.1 Authoring Model

The following observations about temporal media are used to extend the hypertext object and link model:

- The information carried by temporal media can change continuously and there is thus a need to update links dynamically.

- Temporal media are active and can act spontaneously. Any change in circumstances generates *events*.
- There is a requirement for parallelism in the navigating and displaying of multiple temporal media, e.g. displaying video and sound together.

An object oriented framework is used to extend the hypertext model yielding what is termed a *hyperobject* model. The main features of the model are

- *Nodes and links*
  Nodes represent objects stored in an object oriented database. These objects have methods to control their presentation. The model is *open* so that new forms of media can be incorporated.

  Links are stored separately from objects. They take the form

  ```
  <from, conditions, to, message>
  ```

  Examples of links are

  ```
  <avideo, started:30, amusic, play>
  ```

  ```
  <avideo, finished:0, atext, play>
  ```

  These specify that 30 seconds after object `avideo` has been started, message `play` is to be sent to object `amusic` and when object `avideo` has finished, object `atext` is to be started.

  The same methods {`play, stop`} and event conditions {`started:time`, `finished: time`, `clicked`} are provided for objects of every media for consistency.

- *Sub-objects*
  Sub-objects represent *parts* of multimedia objects. Examples are a character in a motion video, a word or paragraph in a text object or a rectangle in a graphic object. These sub-objects can send or receive messages like standard objects.

- *Object groups*
  If links are navigated at the same time then media objects are displayed in parallel. To synchronise other functions such as stopping objects at the same time object groups can be defined. These allow the earliest or latest event to be used to fire a link by using the `min(a,b,c)` or `max(a,b,c)` semantics, where `a`, `b` and `c` are objects in the group.

3.3.2 System Architecture

Timing control among objects is based on events rather than timelines. For example, the internal time of video objects is based on frame numbers. If music is required to start 10 seconds after a video starts, it will still be at the same point even if the video playback speed changes.

The System Architecture is divided into the following three levels:

- *User interface manager*
  Translates interaction into messages and sends them to the link manager or database.
- *Link manager*
  If a link is navigated the source media object sends a message to the link manager.
- *Database*
  The VERSANT database and the C++ programming language are used.

A scenario viewer for graphical specification of objects and links has also been developed.

3.3.3 Evaluation

This model introduces some useful concepts such as objects with methods, and events which can be used to trigger the methods. The concepts of sub-objects and groups are also useful.

The authors admit to the failings of no explicit synchronisation or real-time support for presentation. The system appears to be *stand-alone* in that it seems presentations must run on top of a system which cannot be part of or communicate with other applications.

3.4 CWI

An authoring model, CMIF [**Bulterman91**], and a multimedia support framework, AMF [**Bulterman93**], are being investigated.

3.4.1 CWI Multimedia Interchange Format (CMIF)

A CMIF specification describes a document as a hierarchy of components, each of which is a specific (possibly composite) media type. The hierarchy is traversed either sequentially or in parallel. The order of tree traversal can be changed dynamically in two ways- by hyperlinks or by accessing information non-sequentially, e.g. in reverse. The hierarchy is mapped onto a set of channels which relate the logical components to virtual output devices. Fine grain synchronisation can be specified using synchronisation arcs. At run-time, channels can be active or inactive, influencing document synchronisation.

3.4.2 Amsterdam Multimedia Framework (AMF)

This model attempts to take a specification of a multimedia document and realise it using run-time adaptability. This is achieved by dynamically changing a presentation at run-time based on available resources. An example is a component which can be represented by playing video and audio, playing just audio and a series of pictures or just showing text.

To support applications with such requirements, a framework has been defined which provides partitioning of control concerns in a network infra-structure. The main components are

- *Application Processes (APs)*
  Supply the other components with a specification of object streams, inter-object synchronisation and options used in providing adaptable control.

- *Local Operating Systems (LOSs)*
  Scheduling authority controlling accesses to I/O devices attached to the local workstation.

- *Global Operating Systems (GOSs)*
  Allocates resources on a network-wide basis.

- *Adaptive Information Objects (AIOs)*
  Provides applications with an interface to stored, synthesised, or interactive information. These objects manage the run-time adaptability described above. They provides quality of information rather than quality of service support.

In this model all AIOs are aware of each other's presence and service constraints, along with the options acceptable to the application and supportable by the LOS/GOS. To realise a document, each logical block placed on a channel is associated with an AIO. Interfaces for application control are provided by AIOs.

3.4.3 Evaluation

The CMIF model is a development of the Harmony model. Despite providing a richer set of synchronisation primitives it still is not suitable for applications involving collaboration. It is made clear that the AMF does not solve the multimedia data transfer problems such as guarantees on stream delivery or synchronisation. It only characterises the components in an environment and it indicates their interactions. Individual models to represent the framework functionality need to be implemented.

3.5 University of Geneva

3.5.1 Multimedia Data Modelling

Issues pertaining to the modelling of audio and video (AV) data are discussed in [**Breiteneder92**, **Gibbs93a**]. In this model, data relationships are stored rather than implemented by applications.

Individual elements are defined to be of a certain type, e.g. a sample of type `CDAudioType` or a frame of `JPEGVideoType`.

A *temporal entity type* represents a collection of these and is represented by the form

```
N [ T, Q, D ]
```

- `N` represents the name
- `T` represents the type of composing elements
- `Q` represents the *quality factor*. This is generically expressed, e.g. "broadcast quality" or "VHS quality" for video.
- `D` represents the *discrete time system*. This specifies the interval of specific elements.

An example is

```
VideoEntity[JPEGvideotype, 480x640x8, D30 ]
```

which specifies a video object composed of JPEG frames of 480x640 resolution, 8 pixels deep and 30 frames per second. This scheme is extended in [**Gibbs93a**] as *timed streams* to include other pertinent information.

Derivation relationships provide different views of temporal entity types without the need for replication. They allow the changing of object content, timing and medium type (such as animation to video) or concatenation.

Composition of elements can relate them together spatially or temporally. An example is storing a video with soundtrack as a related object.

[**Gibbs92a**] describes an object-oriented audio and video database framework. It uses the above data model and extends it with the notion that databases should also provide *activities*. This model defines an *AV value* to be a data element and an *AV activity* to be a process that produces and consumes at the value's data rate. These objects can be interconnected to process multimedia data in certain combinations. An *AV database* manages collections of values and activities. This database architecture has not been implemented but an approach based on the use of activities for application construction is described below.

3.5.2 Application Construction

A programming environment for multimedia applications is being developed [**Gibbs92b**, **Gibbs93b**, **de Mey92**]. In this model the media *value* and *activity* terms are used. Activities are implemented as active objects with methods such as `start()`, `stop()`, `pause()` and `resume()`. Active objects have state and methods. They can perform actions even if no message is sent to them and can run concurrently.

A *generic composition graph* is composed of *nodes*, each of which has *connection points*. Connection points can be linked using an *edge*. Such a model is used in the following three ways to allow media objects to build multimedia applications:

- *Data flow composition model*
  This model is concerned with building configurations of hardware and software and controlling data flow between them. A node represents a media activity which can be a source, sink or filter. A connection point represents a *port* and an *edge* represents a connector which exchanges information between ports. Connectors can be shared memory, buffered, delegate or cable, dependent on configuration. An example is connecting the ports of two video source nodes into a video mixer unit which outputs to other objects such as a display.

- *Activity composition model*
  This model describes relationships between temporal sequences. A node represents an *activity*, a connection point represents an *event* and an edge represents a *trigger*. Activities produce and consume events. Example event types supported are the beginning and end of activities. Triggers can relate object control in response to events.

- *Temporal composition model*
  This model is a specialisation of the activity composition model. Object presentation can be related on a timeline, relative to the occurrence of certain events.

A script language has been defined for the combination of activities. Visual composition techniques have investigated specifying this process graphically [**de Mey92**]. An example application, "Virtual Museum", is being constructed to illustrate concepts.

3.5.3 Evaluation

The processing of data using active objects is emphasised in this work which is similar to approaches used in the work presented in this dissertation. However, it is not clear whether using only active objects and configuring them graphically is adequate to specify the functionality of any multimedia application. Little detail is given on problems and methods of implementation but a real application is under construction.

3.6 University of Athens

3.6.1 Description

A platform independent model for the expression of interactive multimedia presentation using a script-based approach is illustrated [**Vazirgiannis93**].

The pre-requisite for such work is a data model that supports the specific attributes and functions of multimedia objects. A model has been partially implemented which is believed to fill these criteria. It stores the data in the form

```
item class [author, format, date of creation]
```

Text, audio, video and images are all supported.

This model supports the notion of *actions*. Unary actions are presentation operations such as image scaling, cropping, filtering, colouring, text formatting, audio cueing, playing, suspending, resuming, stopping, inverting or temporal scaling. *N-ary* actions spatially or temporally affect the presentation of a group of objects. Examples are spatial overlaying, grouping, ungrouping, moving or synchronising.

Events are defined as arbitrary distributed points in time used for the ordering of actions and their coordinated access to shared resources. The following three types of event are handled by this system: the start point of an action, the end point of an action and clicking on predefined button objects. Start and end point events can be used for sequential synchronisation (starting one object after another) or parallel synchronisation (starting two objects at the same time). Buttons can be used for asynchronous interaction.

Scripts specify the structure of composite items which are made up of media items and instructions for their presentation based on events and actions. A composite item has the following components:

- *Spatial data definitions*
  Details of graphics, text, image and button objects can be specified, such as layout parameters. A list of children, composed of other composite objects, can also be defined.
- *Temporal data definitions*
  This section describes temporal data such as audio.
- *Spatio-temporal data definitions*
  Objects involving spatial and temporal concerns, such as video, can also be included.
- *Temporal composition*
  This section is made up of scenario lists which specify presentation and interaction. A scenario list is of the form

  ```
  scenario list <start time, duration, <action>, sync events,
                exception handler, interrupt handler>
  ```

  where `start time` is an expression built up using `and` and `or` operators from absolute time and events. An example start time is `((12.45) or (event1 or event2))`

  `duration` is an expression of the same form as start time specifying when to end.

  `action` refers to a single object (e.g. play, move or display) or to a combination of objects (e.g. overlap, resolve, occlusion, scaling or cropping). Actions can be combined using the $\{\wedge, \vee, *\}$ operators meaning parallel-last (start concurrently, finish when all have finished), parallel-first (start concurrently, finish when the first one finishes) and repetition respectively.

  `sync events` name the START and END events which will be generated at the beginning and end of the `action` part.

  `exception handler` specifies a set of actions to be executed if there is a timeout or a lack of synchronisation.

  `interrupt handler` specifies an action to be performed if the system fails or if the user issues an escape code.

An application example is under development which represents views around the island of Crete using button-based interaction and time offsets to control presentation.

3.6.2 Evaluation

Despite the comprehensive nature of the script facilities provided in this model, the fixed semantics of composite objects could be constraining. This model also lacks the configurability offered by approaches using active objects (as in section 3.5).

Although the model is constrained by only supporting start, end and click events, an attractive feature is that it allows these events to be combined together albeit in a limited way.

Unfortunately no implementation details are discussed and it is not clear whether presentations must run on top of an 'engine' or can be part of or communicate with other applications.

3.7 Emerging Hypermedia Standards

International standards are under development for the representation of multimedia and hypermedia data. The HyTime standard [**ISO/IEC93b**] is designed for the authoring of interchangeable composite multimedia data. The MHEG standard [**ISO/IEC93a**] is designed for the interchange of presentable composite multimedia data. Both support notions of synchronisation and hyperlinking. A good, if early, commentary on the standards is [**Markey91**].

3.7.1 HyTime

HyTime is an application of the SGML standard[1] in which static and dynamic information can be processed and interchanged by hypertext and multimedia applications. The Music Information Processing Standards (MIPS) committee specify "HyTime is a standardised infrastructure for the representation of integrated open hypermedia documents". *Integrated* means that all information is linkable, whether or not it was explicitly prepared for linking. *Open* means link addressing is independent of the file management or network architecture of particular platforms. *Hypermedia* means the union of hypertext and multimedia.

HyTime is represented by a language and underlying model. Application developers use HyTime constructs to design information structures and objects, and then the HyTime language to represent their structure for interchange. The architectural forms and attributes of the HyTime language are grouped into the following six modules:

- *Base module*
  This module is required to support hyperdocument management using SGML.
- *Measurement module*
  This module provides the ability to specify the position and extent of objects in various

[1] For a survey of document standards see [**Brown89**]

virtual time and space measurement units. These must eventually be represented using some coordinate system as a *phase space*.

- *Location address module*
  This module can address objects that it is not possible to address using SGML unique identifiers or that are in external documents.

- *Hyperlinks module*
  There are 3 types of hyperlink in HyTime. *Context links* have 2 link ends, both of which are addressable within the same context, e.g. for footnotes. *Independent links* have 2 or more link ends which are not within the current context; for example they can be references to multiple documents. *Property links* can be used if there is no write access to a document, for example for reviewers' comments.

- *Scheduling module*
  The scheduling module allows 'events' to be scheduled. Their positions can be expressed in terms of their relationships to one another. An example is to show the second video clip after the first completes or the mouse button is clicked.

- *Rendering module*
  When the scheduling module is used there are two ways of controlling rendition- object modification and event projection. *Object modification* allows specification of the order in which objects are to be modified and of the modifiers. Objects can be passed through *alembics* and changed, for example to change the colour map, display a range of video frames at a slower than normal speed, modulate digital audio objects or scale phase space. *Event Projection* is the conversion of events from one phase space to another, e.g. virtual coordinates to real. *Batons* contain *scaling directives* for projection. They can be used for operations such as changing tempo, pause length, or scaling graphic images.

HyTime aims to offer a generic level of support for a variety of applications. Using HyTime attributes, applications can add additional semantics. The model is not intended for encoding internal representation of information on which application programs act while executing.

3.7.2 MHEG

MHEG is a standard for the coded representation of composite multimedia, components of which can be related by synchronisation or hyperlinks. It is targeted at interactive multimedia applications as it is coded to be presented in real-time. The underlying rationale is that standardisation at the monomedia level is not enough to guarantee application portability.

MHEG uses object-oriented concepts yielding objects which are active, autonomous and reusable. Data is represented as MHEG objects which are coded with the aim of direct presentation. Internal details are masked and applications must only deal with high level functions. To realise MHEG presentations requires a tool described as an MHEG *engine*.

There is a distinction between the concept of an object and that of a *viewer*. Objects contain reusable structural information. A viewer is a specific view of one of these objects used in some presentation context. A viewer is not an interchanged MHEG object but a reference to a presentable object used by the composite object to view its components.

The MHEG standard defines the following set of classes, instances of which correspond to interchanged objects:

- *Action class*
  Actions allow an MHEG engine to affect the state and presentation of objects and viewers. Preparation actions change the state of underlying MHEG objects. Presentation actions change the state of viewers. Projection actions affect projection of viewers, e.g. volume for audio or dynamic rendition effects. Interaction actions can be used to change the result of interaction such as using the same user selection for different purposes.

- *Link class*
  An MHEG link is a general mechanism to describe relations between objects and viewers in time, space or conditionally (e.g. if a button object is pushed, an audio segment object plays).

- *Script class*
  By using the script class, complex inter-object links can be specified. A script object is represented by an external script and a type attribute to specify which language it is written in. It is assumed that the script text can reference objects and viewers.

- *Content class*
  Objects of this class are represented as encoded information and a *hook* attribute which specifies information required for presentation such as encoding, data formatting and other parameters.

- *Selection class*
  Objects of this class provide a tree of allowable selections by the user.

- *Modification class*
  Objects of this class reference content objects which are to be modified.

- *Composite class*
  Objects of this class allow content objects to be related together with links, actions and scripts and can describe the presentation of different viewers. Such composite objects provide the support for data interchange.

- *Descriptor class*
  Based on information of a set of objects, instances of this class recommend the minimal resources required for presentation such as minimum memory and CPU requirements.

- *Macro class*
  Objects of this class allow the instantiation of complex structures.

3.7.3 Evaluation

The design of these standards must embrace the facilities of a wide range of computing systems and environments. They emphasise information exchange and must encapsulate all related data. They are thus not suitable for developing collaborative applications or using facilities in distributed environments such as cameras.

The standards are compatible in that HyTime is designed for authoring and is not suitable for real-time rendering of multimedia data whereas MHEG has a minimal capacity for document revision and is not good for highly interactive authoring.

3.8 Bellcore Information Networking Research Laboratory

Touring machine [**Bellcore93**] is an application development platform which manages multimedia communications, routeing, resource management, presentation control, device control and media synchronisation for communication-based applications. It separates the management policy (in the form of an Application Programming Interface (API)) from the mechanism that enforces it.

3.8.1 Application Programming Interface (API)

The API defines the user's interface to the Touring machine system. It provides control over

- *Client registration*
  A client must first register at a station. The system authenticates a client and allows his/her physical location to be known. The client specifies and can change a set of endpoints, privacy details and other attributes. An example is `Register bob` with

  ```
  camera: video source;
  monitor: video sink;
  mic: audio source;
  speaker: audio sink;
  ```

  and `set privacy all`.

- *Session management*
  Facilities are provided to configure topologies and policies involving many clients. The following two phases are involved: *negotiation* which tries to allocate necessary trunks and bridges and *allocation of resources* which sets up session functionality. A three user audiovisual conference would require two *connectors* to be created- a video one and an audio one, to manage connection of relevant sources and sinks. Dynamic reconfiguration is possible such as changing a source endpoint from a camera to a file.

A *name server* can be queried to find out certain information such as on-going sessions and users involved. Direct client to client signalling is possible.

3.8.2 Software Architecture

A series of distributed objects work together to realise the functionality of the API. Objects can be *permanent* or *transient*. They communicate by asynchronous message passing. The infra-structure is divided into the following two levels:

- *Session control*
  This provides logical control of communications sessions. *Station objects* provide an interface between clients and the infra-structure. They verify API requests and route them to an appropriate server. They also provide sharing of network access ports. *Session objects* are transient objects created by the station object to manage a session.

- *Transport control*
  This provides resource allocation and network transparency. The *resource manager* manages allocation and deallocation of resources such as trunks and bridges. It is optimistic, using approximate values for heuristic routeing. *Transport objects* are instantiated by the resource manager and track allocation and deallocation for the session. *Resource objects* manage sets of physical resources. They communicate directly with network hardware such as switches, bridges and trunks.

The name server maps queries into an SQL-like form which is implemented by its *datalayer*.

Example applications have been developed using this system. *Cruiser* [**Fish93**] is a teleconferencing system for informal communications between co-workers. *Rendezvous* [**Hill93**] provides shared views of information through synchronous communication, such as a multi-user whiteboard.

3.8.3 Evaluation

This system provides an excellent example of a platform functioning in a real environment. It is biased towards communication-based applications and is thus lacking in facilities for interactive presentation.

3.9 University of Lancaster

Abstractions have been built over the Lancaster multimedia environment to make programming easier [**Coulson93**]. Higher level services for synchronisation [**Coulson91**] and session configuration [**Brooks94**] have also been constructed. This work attempts to extend existing ODP standards for multimedia.

3.9.1 Programming Abstractions

The following three types of abstraction have been built to simplify the programming of multimedia applications:

- *Devices*
  Device objects represent sources, sinks or transformers of multimedia data. Multimedia data is represented as *chains* which are a collection of pointers to media specific data. Operations can be performed on them such as concatenation. Device objects use

endpoints for input and output. Endpoints have a specific QoS specifying the data characteristic (e.g. RGB_video, PCM_audio) and the rate at which data can be presented to or removed from them.

- *Streams*
  Streams represent an abstraction over a transport protocol and connect together device endpoints. They are unidirectional and have associated QoS attributes such as linktype, maximum throughput, jitter, latency, bit error rate and minimum utilisation.

- *Groups*
  Groups provide group invocation and group communication for continuous media. A named group of a certain type can be created, and can then be controlled as if it were a single object. It can also be connected to other objects in one operation.

The system supports *back-call* operations so that an application can be informed asynchronously of the occurrence of certain events.

3.9.2 Synchronisation

A synchronisation service provides both event-based and continuous synchronisation. Event-based synchronisation allows the performance of actions at certain points in a stream. Requirements are assumed to be specified using a form of script. Continuous synchronisation aligns the presentation of streams such as related video and audio. The realisation of this process is termed *orchestration* and is managed by orchestration services.

3.9.3 Configuration

A service is outlined in [**Coulson93**] and extended in [**Brooks94**] that allows device objects to be configured using fewer programming statements than are required to configure objects directly. A collection of device objects can be connected automatically at a specified level of QoS. Existing configurations can be modified on-the-fly.

3.9.4 Evaluation

The abstractions and services developed in this work greatly simplify programming within multimedia environments. They take into account environmental factors such as synchronisation and QoS. The configuration service is similar to concepts presented later in this dissertation.

No specific authoring model is proposed to simplify the use of services such as configuration and synchronisation. This is assumed to be the concern of a higher level script language.

3.10 Analysis and Conclusions

This chapter has reviewed work relating to the authoring and presentation of multimedia applications. The analysis below comments on points raised in the above work with relation to the research documented in this dissertation.

- *Data modelling*
  A data model should provide a method of storing references to multimedia data items since it is not practical to reference them on a per application basis. It must be possible to manipulate data separately from its run-time processing (sections 3.2, 3.5). A data model must take into account the fact that in a distributed multimedia environment data can be of different medium types and formats and stored on different forms of device. It can be useful for a user to store other instance specific information with data objects.

- *Active objects and configuration*
  Active objects represent a powerful and uniform mechanism for device access and processing of multimedia data (sections 3.5, 3.9). These objects are controlled by interfaces. They can be connected in configurations, and special services can be used to simplify this process (section 3.9). An active object must have knowledge of the format which it manipulates. Such complexities should be abstracted from the user wherever possible.

- *Support for events*
  It must be possible to initiate specific actions on the occurrence of events (sections 3.1, 3.3, 3.5, 3.6, 3.7). In the models described, events are limited to the start and finish of the presentation of particular objects and, in section 3.6, the pressing of buttons. The back call mechanism in section 3.9 can be used to build generic event capabilities but requires higher level event marking and handling functions. The simple event combination facility outlined in section 3.6 is an attractive feature. If expanded to allow any combination of events it can represent a powerful application development mechanism.

- *Presentation services*
  Services to manage aspects of the presentation process, such as the synchronisation service in section 3.9, are attractive since they remove complexity from applications.

- *Programming language support*
  Specially developed script languages (sections 3.6, 3.7.1) are attractive since standard languages are often inappropriate for representing presentational aspects of multimedia applications. However, it is important that such approaches can be made part of an application developed using traditional techniques. They should not try and represent the full functionality of a programming language.

- *Range of applications supported*
  The authoring approaches represented in the above models are limited to certain classes of application. For example the approaches in sections 3.1, 3.3, 3.4 and 3.6 are only appropriate for compositional applications such as hypermedia documents. The approach in section 3.8 is tailored more to the construction of conferencing-based applications.

  Aspects of the models represented in sections 3.5 and 3.9 could be used to construct many classes of application but require high level services and an authoring approach to simplify construction for the application developer.

In summary authoring and presentation support for multimedia applications must be *open* and *extensible*. It must take into account the complexities of real environments and not provide a subset of facilities to realise authored applications. With regard to data modelling, it must be possible to extend the types of medium and format supported. It must also be possible for an application developer to tailor individual data objects. The range of active object abstractions must be extensible to support new devices and processing functions. It must be possible to support any form of event and invoke operations within an application on their occurrence. Facilities should be capable of constructing a wide range of applications. It must be possible to add new system components to support emerging applications.

Chapter 4

Authoring Model

This chapter describes authoring techniques which allow an application developer to express interactive presentation of multimedia. The authoring model promotes the principles of

- *Generality* - Providing features which are not tailored to any particular environment or to any particular application requirements.
- *Reusability* - Providing facilities that are developed to be useful to users in the future instead of providing support on a per application basis.
- *Tailorability* - Making it possible to configure the facilities to provide functionality for particular applications.
- *Extensibility* - Allowing desirable features which are not currently available within the platform to be implemented. Any new support which is added can be utilised through the same authoring framework.
- *Openness* - Ensuring the model is not *stand-alone* in that authored presentations can be made *part* of an application program.

4.1 Overview of Authoring Facilities

The authoring model summarises the facilities perceived to be necessary to enable development of a wide range of applications within multimedia environments. In this chapter, the discussion of authoring concepts is integrated with techniques for their specification. The framework described has been implemented as part of the IMP platform, introduced in section 1.3. The two following chapters outline how an interactive presentation, authored using this framework, is realised at run-time. In this chapter only services with which the application developer is directly involved will be described.

The discussion of the authoring process has been divided into the following three main parts:

- *Multimedia data modelling*
  Section 4.2 describes facilities which allow an application developer to reference distributed multimedia data items as *data objects*. Instances of these can be tailored by the addition of marked events and attributes.

- *Expressing interactive presentation*
  A language enabling the expression of interactive presentation using these data objects is described in section 4.3.

- *The controlling application*
  Section 4.4 describes how the interactive presentation specified in scripts can be included as part of user applications. Many such presentations can be controlled by cooperative applications.

4.2 Multimedia Data Modelling

The data model is designed to allow the representation of multimedia data items as objects, which can be used without explicit knowledge of environmental details. The data model allows *instances* of *data objects* to be created. A data object represents an encapsulation of a reference to a multimedia data item. It can be tailored by marking parts of the contents (events of interest) and annotation with attributes. Figure 4.1 illustrates the video data object "Opera" which represents some footage of a meeting of the 'Opera' group[1].

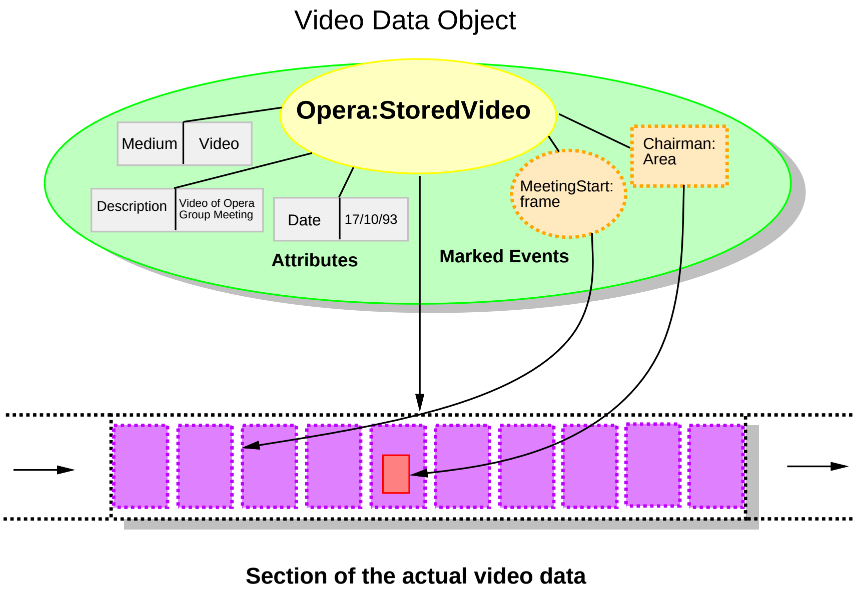


Figure 4.1: Illustration of a data object

4.2.1 The Aims of the Data Model

Within a multimedia environment, data can exist in many heterogeneous forms. Data items can be of different medium types and either continuous or still in nature. Items of the same medium may differ in their encodings or originate from different types of devices. For some

[1] Multimedia research group within the University of Cambridge Computer Laboratory

medium types, such as video, devices (such as a camera) can produce live data and others (such as a storage service) can retrieve previously stored data.

It is a requirement to be able to present such heterogeneous data items within the same application. It is useful to keep homogeneous persistent references to data items due to the differing forms of the underlying references and the unnecessary expense of referencing data items on a per application basis. It is desirable to mask the complexity of the data environment from the application developer. He/she should only be concerned with generic and relevant characteristics of data items such as their medium type which dictates the manner in which they can be manipulated. In figure 4.1 the data object is referencing a stored video object. The object format and device the video is stored on are irrelevant. The platform manages the complexity involved in referencing (and presentation) for the application developer.

4.2.2 Data Modelling Scheme

The data model provides the user with named data objects. A data modelling interface is provided to allow application developers to create, manipulate and relate objects. Object instances can be tailored in ways shaped by their medium type. The data modelling interface is realised by a presentation database system described in chapter 5. This provides persistent storage and manipulation of objects via interfaces. The presentation database promotes reuse as it acts as a repository from which objects can be selected for particular applications. Specific tools can be developed to manipulate aspects of the data modelling process. The data modelling interface has been designed to be generic so that a wide range of tools can be implemented above it.

The main features of the data model will be described- how objects are added into the system, how they are manipulated and how new instances can be created and specialised by the marking of events and attributes.

4.2.3 Adding Data Objects

The data model attempts to mask out environmental complexity and provide a uniform setting for data manipulation. However, the system must initially be given details about objects so it knows how to process them. The environment services used as examples in this section are amongst those introduced in section 2.2.

When a new object is created the system requires the following information:

- *Object name*
  An identifier for the object by which it can be retrieved.

- *Medium type*
  Identifies the nature of the object's medium. For example `Text`, `Picture`, `Video`, `Audio` or `AudioVideo`[2].

[2] Some continuous media formats and devices allow the interleaving of video and audio information. These can be represented as a single medium type.

- *Format type*
  Different formats of the same medium type can be supported by the system. The user must specify which particular format the object data is stored in. For example, for medium type `Video`, formats `PandoraVideo`, `JPEGVideo` or `MPEGVideo` could be selected. For medium type `Picture`, formats `GIF`, `TIFF`, `XBitmap`, `SunRaster`, `JPEG`, `PM`, `PBM` or `Postscript` could be selected.

- *Source device type*
  A reference to the type of device on which the object has its source. Examples are `PandoraCamera`, `VideoRepository`, `NFS` or `MSSA`.

- *Location data*
  Associated with the source device type is a template for location data which must be provided. For example, for device type `NFS` a domain pathname must be provided or for `MSSA` an SSID.

The database system must check information given by a user to ensure that the references are valid. For example, medium type `Text` cannot be supported on device type `VideoRepository` and medium type `Audio` cannot have format type `Postscript`. In the case of a stored object, the system can check that the object exists on the referenced service (despite this it can be later deleted leaving a *dangling reference*).

Once an object has been created, the application developer need not reference its system specific details. The database is an open environment in that support for new types of medium, format, device or event can be added. Further details of this nature are provided in later chapters.

4.2.4 Manipulating Objects

Once objects have been added, they can be manipulated by name. Operations available on them include

- *Tailoring*
  Objects can be tailored by the marking of events and annotation of attributes. These operations are described in greater detail below.

- *Cloning*
  Application developers may require different instances of objects which reference the same data item in different application scenarios. For this purpose, objects can be cloned to produce new named objects. Cloned objects inherit all the type, event and attribute information associated with their parents. They are fully independent and can be further tailored.

- *Querying of type information*
  Tools to manipulate the data need to know the type information associated with the object. System specific details can thus be accessed if required.

- *Deletion*
  Individual events and attributes can be deleted from objects. Entire objects can also be deleted.

4.2.5 Marking Events

Multimedia applications are often event-driven, i.e. asynchronous occurrences, such as stages in a presentation being reached or user interaction, can cause further actions to be initiated. Many of these events are associated with the presentation of or interaction with data objects within a presentation. It is therefore desirable to be able to create instances of data objects for specific presentations, marked with the events required for run-time.

Types of event which can be marked on particular data objects are dictated by the medium type of that object. Types of event can be divided into

- *Content interaction*
  Events caused by the user interacting with the data in some way. Static media have no temporal characteristics and thus these are the only events which apply to them. They involve marking part of the content of the data, for example in a hypertext application, certain words of text are marked as *hot spots*. Clicking on a marked word generates an event which can be used to trigger the initiation of actions. Content interaction events can also be associated with media such as pictures and video. Areas of pictures or video can be marked so that if the user clicks on them events are generated.

- *Temporal occurrences*
  These are events caused by occurrences at certain points in a continuous medium stream. For example, a particular frame of a video object can be marked. When the frame is presented at run-time, interested parties are informed. An application can initiate some actions based on this event, such as the display of a related text subtitle. More complex temporal occurrences can be specified such as notification if a certain word is detected in an audio speech sequence or if a 'red motorcycle' is detected in a video sequence.

- *Temporally-dependent content interaction*
  Complex processing functions can allow content interaction to be dependent on temporal occurrences within continuous media. An example is specifying an object in a video clip, the movements of which are to be tracked through subsequent video frames. Events could be notified if it is clicked on.

The act of informing interested parties of a run-time event occurrence is termed *event notification* (see section 4.3.3.1). Within the data model, an application builder can *mark* events, the occurrence of which will be notified at run-time to any application using the data object. Figure 4.1 illustrates two types of marked event. `MeetingStart` is a frame number at which the meeting starts in the video. `Chairman` marks the area occupied by the chairman as a clickable hot spot.

To mark an event the following information must be specified:

- *Event name*
  Marked events are named. This allows them to be retrieved and manipulated by name. It also means that an application can refer to them by name in event handling statements (see section 4.3.3.4).

- *Event type*
  The *types* of event that can be marked on a particular object is dictated by its medium type. For example, events for video objects can be marked frames or areas of the screen which can be clicked on; events for text can be hypertext links.

- *Type-specific data*
  Each event type has related parameters which must be provided to describe an event fully. For example with video event type `MarkedFrame` a number must be provided to reference the frame to be marked. With event type `ClickArea`, which monitors an area of a window for click events, 4 coordinates, describing the spatial area of interest, must be provided. The application developer does not necessarily have to access type-specific data directly, but can use a tool as described in section 4.2.8.

Events marked in the data model are all *primitive* in that they just describe a single event occurrence. Methods for composing such primitive events to enable the monitoring of complex scenarios are an aspect of the design of the script language described later in section 4.3.3.5. Other types of event can be of interest to the application developer, such as occurrences within an environment or temporal delays. These are described in sections 4.3.3.2 and 4.3.3.7.

4.2.6 Annotating Attributes

Application developers can associate attributes with data objects. Such attributes are composed of a two-field tuple of the form {`attr name, value`}. Attributes are entirely user-defined and have no system-defined semantic meaning. Envisaged uses of attributes are

- *Filing*
  A user may simply want to store information about particular instances of stored objects such as a description of the contents of the data or the date it was created.

- *Indexed retrieval*
  An application developer creating an application involving interactive presentation can browse a catalogue of data objects. If only rough requirements are known, he/she can search for objects using certain attributes. For example, if he/she requires a video component involving motorcycles, then a query can be used where `subject = motorcycling` and `medium = video`. Maintenance and policy for such attributes is up to database maintainers.

- *Controlling by characteristic*
  It is also envisaged that an application may require details of certain characteristics regarding the data objects it is using. For example, if the application needs to degrade the resolution and frame rate of all video objects, it can perform a query to find out which of the objects it is using are video.

Named attributes can be added to data objects regardless of type. The value of attributes can be changed at any time. Figure 4.1 illustrates attributes which may be used for any of the above functions.

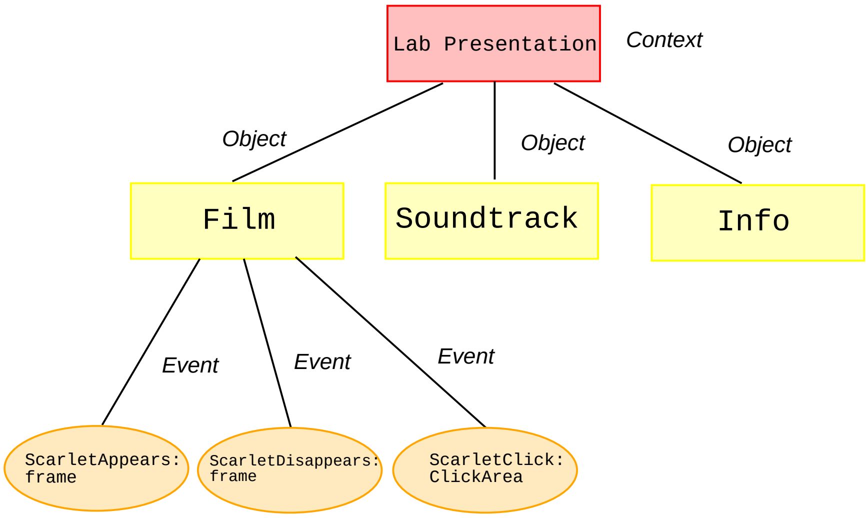


Figure 4.2: Part of the Lab presentation context

4.2.7 Contexts

Data objects can be collected together as composite objects referred to as *contexts*. The purposes of contexts are

- *Separation of database state*
  Separation of the database into sub-collections is conceptually like separating a file space into directories, where related objects are kept together.

- *Representing data for presentations*
  To higher levels, contexts have a semantic purpose in that they represent the data to which a script, specifying the run-time behaviour of a presentation, can be applied.

Part of a context "Lab presentation" is illustrated in figure 4.2. Among its contents are a video object called `Film`, an audio object called `Soundtrack` and a text object called `Info`. `Film` has the following marked events: `ScarletAppears` which signifies the frame at which Scarlet, a character in the video, appears on screen; `ScarletDisappears` which signifies the frame at which she ceases to be on-screen; and `ScarletClick`, a `ClickArea` event, which marks as a hot spot the part of the screen she occupies whilst on it. This context is used to build an example in section 4.3.

4.2.8 Manipulating the Data Model

The database has an object-oriented interface to promote an open authoring approach, i.e. any tool can be put above it to manipulate the data model (see figure 4.3). Some useful tool functions are outlined below.

- *Graphical object modelling and browsing*
  A prototype tool, *WIMP*, has been implemented to provide graphical manipulation of

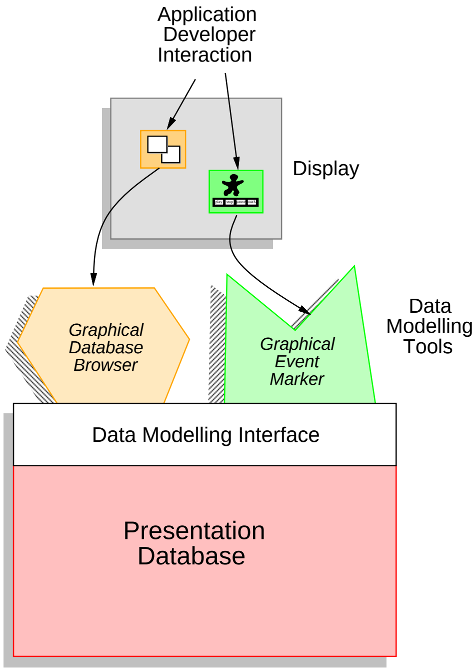


Figure 4.3: Data modelling tools

object data. A series of forms can be called up and filled in to add objects. Catalogues of objects can be viewed to see if they are appropriate for particular applications. New instances of objects can be created and specialised by the marking of events and attributes. Queries can be used to retrieve desired classes of objects. This tool is illustrated in chapter 7.

- *Event marking via direct manipulation*
  Tools are envisaged with which an application developer can view particular data objects and, using direct manipulation, mark events on them. An example is a tool which the application developer could use to play and pause video at certain points, and mark particular image details or frames.

4.3 Expressing Interactive Presentation

This section describes techniques to allow an application developer to express interactive presentation of modelled data objects. A script language has been designed for this purpose since conventional languages are not explicitly suited to the task. To manage a presentation, the script language provides constructs to control the following functions:

- *Referencing presentation data*
  A way of specifying the modelled data which is to be presented is described in section 4.3.1.

- *Data presentation*
  This involves taking the data from devices, manipulating and displaying it. Configura-

tions of multimedia *presentation objects* are specified for this purpose. This process is discussed in section 4.3.2.

- *Event handling*
  Event-driven rules allow the occurrence of one or a certain combination of events to influence the presentation process. Event handling is outlined in section 4.3.3.

- *Manipulation of groups of objects*
  Objects can be related together as *object sets*. These enable group invocations to be performed. It is often useful for groups of objects to behave in the same way, e.g. defining policies to adjust QoS by degrading groups of objects. It is also possible to use object sets to relate objects for presentation synchronisation (e.g. for lip-sync'ing video and audio objects). Section 4.3.4 describes these facilities in greater detail.

4.3.1 Referencing Presentation Data

An application developer collects data required for a presentation in the form of a context (see section 4.2.7). Using a script, the presentation of individual objects from within this context can be described. The first line of a script always references the context to which it must be applied. A script operating on the Laboratory presentation context would start

```
context LabPresentation;
```

A script can be defined as a template and then instantiated with actual values before it is activated. Consider an application which implements a *video phone* which connects two users in a video and audio conference. The same script structure is required regardless of callers. Only the names of objects involved (the live sources of audio and video) will change. The following script template can be defined for this purpose:

```
context videophone(caller_audio, caller_video,
                   receiver_audio, receiver_video);
```

The four parameters can be used throughout the script to represent actual data objects. The script can be instantiated with the real values before the start of a conference session. If Jean wanted to call John, then the data objects could be `JeanAudio`, `JeanVideo`, `JohnAudio` and `JohnVideo`. Instantiation of these parameters is described further in section 4.4.1.

4.3.2 Data Presentation

In different applications, data presentation requirements may be markedly different. These requirements can be realised by defining an appropriate configuration of presentation objects using the script language.

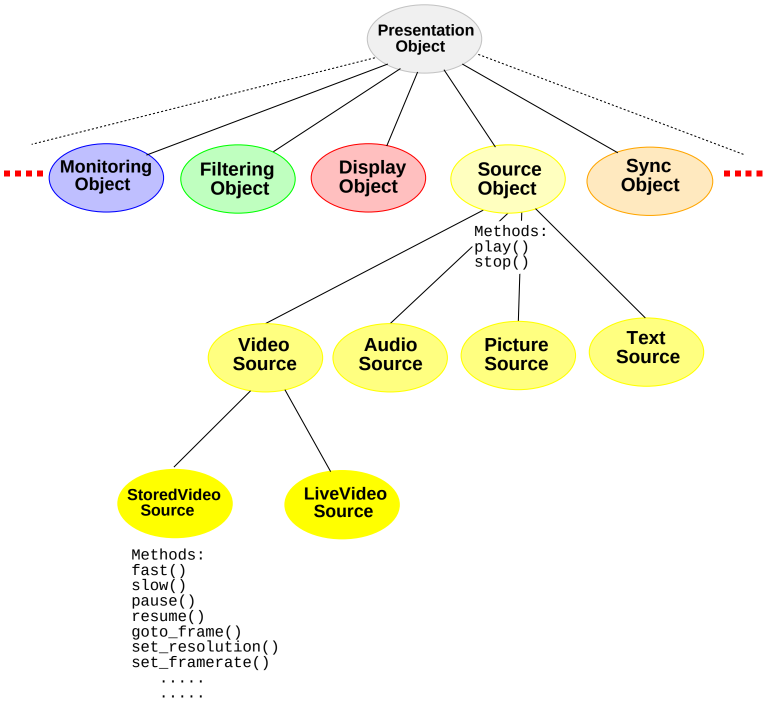


Figure 4.4: Application developer's view of a presentation object hierarchy

4.3.2.1 Presentation Objects

A presentation object is an encapsulation of a presentation function for a particular medium type. Such objects input data, perform the relevant function, and output the data. For continuous media this is a continuous process. Presentation objects may also examine the data for certain types of event. They can be controlled dynamically using script facilities.

Presentation objects promote the concept of reuse by defining a series of standard abstractions. The choice of objects is represented as a class hierarchy. Part of such a class hierarchy is illustrated in figure 4.4. This example shows how all presentation objects are derived from class `presentation_object`. Each presentation object has a class and associated methods. Sub-classes can have additional methods to the ones they inherit from their parent, such as the stored video source object in the diagram. Presentation objects perform one or more of the following functions:

- *Monitoring*
  Examining the data for particular phenomena such as a word of a text object being clicked on or a certain video frame being presented.

- *Analysing*
  Parsing the data, e.g. analysing speech for recognisable patterns or looking for someone's face in a video clip.

- *Filtering*
  Taking the data and changing it in some way. Examples are improving the quality of audio data or performing dynamic visual effects on video.

- *Displaying*
  Displaying the data in a certain style. Display components associated with the same presentation can be shown within the same *workspace*, i.e. a window into which the display component can be laid out. Figure 4.5 illustrates an example workspace. The characteristics of a workspace are determined when it is created (see section 4.4.1).

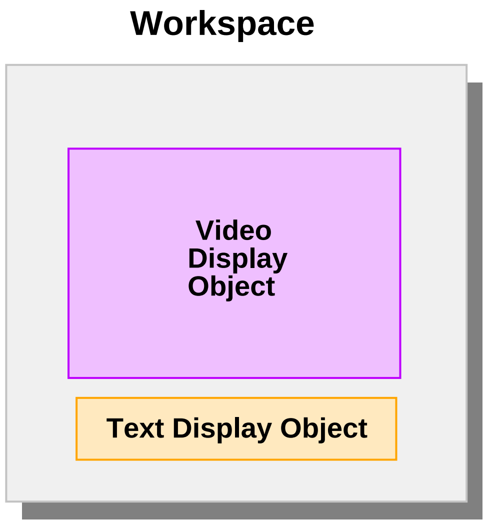


Figure 4.5: An example workspace containing display components

- *Device access*
  Source objects can take data from a device (either a live device or a storage service). Sink objects can be used for putting data onto storage devices.

- *Synchronising*
  For continuous media, intra-stream synchronisation can be performed to remove jitter.

From a library of presentation objects, application developers can select objects they wish to use in a presentation.

4.3.2.2 Presentation Object Configurations

Configurations promote the notion of tailorability by allowing data objects to be presented by means of a user-defined combination of presentation objects. Within a configuration, the information associated with a data object is passed between presentation objects. The output of one presentation object is the input of the next. In configuration definitions, some presentation objects must be specified explicitly and some are inserted automatically.

To specify a configuration of presentation objects the following syntax is used:

`SourceObject | PresObject`$_1$ `....  | PresObject`$_n$ $>$ `SinkObject`

`SourceObject` is the name of a data object within the current context which represents the source of data for the configuration. A source presentation object is created to remove

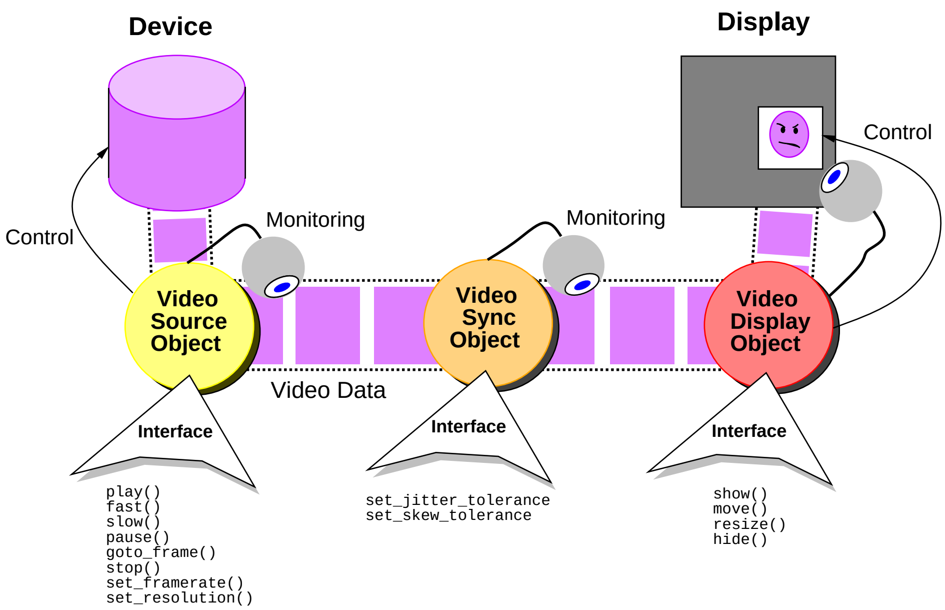


Figure 4.6: Application developer's view of a configuration

the data from its device. Each `PresObject` is a presentation object class. A presentation object instance for each class is created. `SinkObject` is the name of a data object which is created within the context representing the data outputted from the configuration. A sink presentation object is created to manage this process. The input of each presentation object in the configuration is linked to the output of the object preceding it. Every configuration must have one `SourceObject`, zero or more `PresObject` and zero or more `SinkObject`. In some situations, the system can insert presentation objects automatically into configurations.

This can be illustrated with an example. To display the video data object `Film` using an object of class `VideoDisplay`, the following statement could be used:

```
Film | VideoDisplay ;
```

For all configurations the system automatically creates a source presentation object to access the source data. In this case it will create a source video object to access the data associated with `Film`. The source and display objects are controlled separately but act on the same data. The system may create other objects and automatically insert them, such as objects for intra-stream synchronisation. These objects remove jitter before display, thus in this case it is inserted before the display object. The application developer's view of this example configuration is illustrated in figure 4.6.

The choice of presentation objects to be used in a configuration should be based on

- *Presentation requirements*
  A configuration should be designed on the basis of processing, analysis and display requirements for a particular data object.
- *Event monitoring requirements*
  Presentation objects manage event monitoring at run-time, i.e. the process of checking

whether any marked events have occurred. For example, synchronisation objects can monitor for frame boundary events and display objects can monitor for events associated with user interaction. To ensure that required marked events are monitored for, relevant presentation objects should be included in a configuration. In some cases this is done automatically (as with the synchronisation object).

Details of the functionality of all presentation object classes must thus be available to enable the application developer to select appropriate objects. These details include presentation function, available methods and event monitoring capabilities. The situations when presentation objects are automatically inserted into configurations must also be known.

4.3.2.3 Controlling Presentation Objects

At run-time, presentation objects can be controlled dynamically by invoking their methods. Although the realisation of presentation objects is described in chapter 5, a factor associated with the principle of generality must be pointed out at this stage. Defining an authoring approach using generic abstractions must attempt to select generalised functional requirements of individual media, regardless of system-specific details. There is no guarantee that the underlying system support is available or that all functions are even possible using some systems. Thus, although a standard set of methods is provided for each presentation object, it is sometimes not possible for all of them to be realised.

In the above case the configuration built around the video data object `Film` has a source object with methods {`play`, `fast`, `slow`, `pause`, `resume`, `goto_frame`, `stop`, `set_framerate`, `set_resolution....`}. The video display object has methods {`show, move, resize, hide ....`}. The application developer is aware of objects which the system automatically inserts and can control them also. In this example, the synchronisation object has methods {`set_jitter_tolerance, set_skew_tolerance ....`}.

A method invocation has the following parts:

- *The processing configuration*
  This has the same name as the data object the configuration is designed to process.
- *The presentation object(s)*
  These can be referenced by class or by class index. If a class reference is used, then the invocation is performed on every object in the configuration that is of the referenced class or is a sub-class of it. A class index specifies numerically a member of the configuration.
- *The method to invoke*
- *The method parameters*

As an example, the video will be paused and its window moved to a 50 by 50 area in the top left corner of the parent window. Invocations of the following form can be used:

```
Film$Source.pause();
Film$Display.move(0,0);
Film$Display.resize(50,50);
```

In this example the presentation objects are referenced by class. The invocations specify that the `pause` method should be invoked on all objects derived from the class `Source` in the `Film` configuration. Also the `move` and `resize` methods should be invoked on all objects which are derived from the class `Display`. Using the class index method, a specific instance of a particular class in the configuration is referenced. For example, assume there are two components in the configuration of class `VideoDisplay`. The `move` instruction can be invoked on the first one using the following statement[3]:

```
Film$VideoDisplay[0].move(0,0);
```

Presentation objects have methods not only to affect the flow of information such as `pause` or `fast`, but also to affect the quality of information presentation such as `set_resolution` or `set_framerate`. These *quality control* methods are useful in decreasing the amount of resources used by particular objects. Lower resolution and frame rate for a video object, for example, imply less network bandwidth and fewer processor cycles in processing and display.

4.3.3 Event Handling

Multimedia applications may be event-driven rather than sequential in nature. Asynchronous occurrences at run-time such as user interaction or system signals can be used as triggers for actions to be initiated. Section 4.2.5 described how events can be marked on data objects. If these objects are used as part of a presentation, then at run-time any occurrence of their marked events will be notified (see section 4.3.3.1). Events generated by the system and external applications may also be notified. The script provides event rules to specify actions to perform on the occurrence of events.

4.3.3.1 Event Notification

An event notification is the act of an event monitoring process informing an interested party of the run-time occurrence of a marked event. Figure 4.7 illustrates event notification with reference to the Laboratory presentation context. In the example, object `Film` is illustrated. This has three marked events- `ScarletAppears` and `ScarletDisappears` which are both marked frames of the video data, and `ScarletClick` which is an area of the screen to be clicked on. If an event occurs at run-time, it is *notified* to a controlling application (in this case an active context - see section 4.4.1). In the case of the example, `ScarletAppears` and `ScarletDisappears` are notified when their corresponding marked frame is presented. `ScarletClick` is notified if its corresponding area of the display is clicked on.

A statement of the form `object.event` is used to reference the notification of an event occurrence in the script. For example `Film.ScarletAppears`. The notification of marked events can be used to trigger invocations using event-driven rules (see section 4.3.3.4).

4.3.3.2 System Events

An application developer often needs to be aware of events occurring in the environment. *System events* are associated with objects of particular medium types by default. Examples

[3] Indexes start at zero

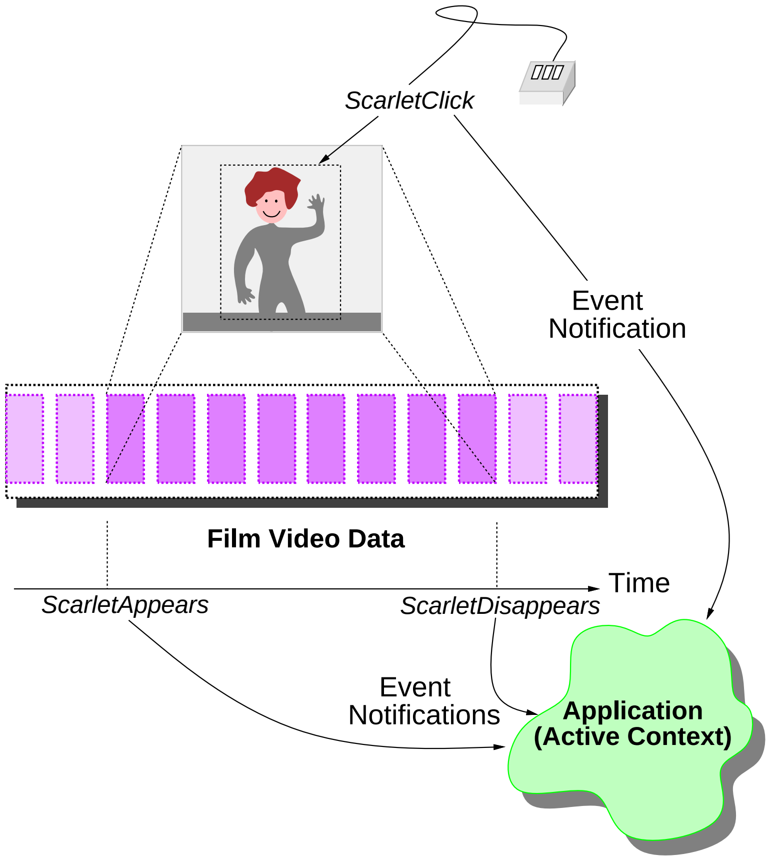


Figure 4.7: Event notification in Laboratory presentation

are system events to signal failures to gain necessary resources or to indicate resource renegotiation is required. Objects of type `StoredVideo` can have default marked events such as `DeviceNegotiationFailure` or `RenegotiateNetworkBandwidth`. The ability to detect such system events allows an application developer to take appropriate action, such as adjusting presentation object resource usage.

4.3.3.3 External and Default Events

It can be desirable to perform certain operations in response to events signalled from external applications. *External events*, i.e. events not associated with objects in the context, can be defined within a script. An example is an event which will be notified from an external application to inform a presentation that it cannot maintain its current level of QoS. In this case the external application could require resources tied up by this presentation. Both could have been started up by the same user. Rather than let the system decide on allocation, the user's other application can use an external event to trigger a rule to degrade resources in the same way as a system event would. The statement below specifies that an event called `QoSEvent` is to be recognised within the script for this purpose.

```
external QoSEvent;
```

This event can be treated in the same way as those marked on data objects. To refer to the notification of such an event in a script requires that its name is used, in this case `QoSEvent`. Using the context tools interface (see section 4.4.3) external applications can notify that such events have occurred.

External events have no defined semantics. They are merely events notified from external applications using an external event name. It is also possible to include some *default events* with pre-defined semantics. The notification of default events is built into the system and they do not have to be declared as `external`.

`any` is an example of such a built-in event. It occurs every time any other event is notified (and is illustrated further in section 4.3.3.5).

4.3.3.4 Event-Driven Rules

Event driven rules specify one or a combination of events which, if *detected*, cause the rule to *fire*. This triggers a series of invocations on presentation objects. An event-driven rule is specified using the form

```
rule <name> = <event expression> -> { <list of invocations> }
```

- *List of invocations*
  A list of invocations on any of the objects in the context (of the form illustrated in section 4.3.2.3).

- *Event expression*
  An event expression specifies one or a combination of primitive events. Events can be those associated with data objects or those specified to occur from outside the system (as described in section 4.3.3.3). As an example of an entire rule, it can be specified that when the event `ScarletDisappears`, associated with the object `Film`, occurs the presentation of object `Film` should go back to the beginning.

  ```
  rule restart = Film.ScarletDisappears -> {
                                              Film$Source.goto_frame(0);
                                            }
  ```

  An event expression may involve the combination of primitive events using certain operators to yield complex monitoring scenarios. An approach from the field of active databases has been employed [**Gehani92**] which is ideally suited to this form of application. A *composite event expression* allows any possible combination of events to be described. As an example, a scenario with relation to object `Film` will be specified (at this point assume all objects have been configured). If the area where Scarlet appears is clicked on between the events which indicate she is on screen then pause the video and display the text information.

```
rule IdentifyScarlet =

  and(inbetween(Film.ScarletAppears,
                Film.ScarletDisappears)
      Film.ScarletClick)

        ->
              {
                Film$Source.pause();
                Info$Display.show(80,270,250,80);
                Info$Source.play();
              }
```

The event expression `inbetween` is a user-defined expression (see section 4.3.3.6) which is detected if an event occurs after an event of the type in the first parameter but before an event of the type in the second parameter. The `and` expression is used to specify interest only if the `inbetween` event is a `ScarletClick`. A full explanation of composite event operators is given below.

- *Rule name*
  Each rule is named. The rule name can be incorporated into the event expressions of other rules. This has the effect notionally of copying the event expression of the referenced rule into the new expression at the point it is referenced. Rules are also named so that external applications can be informed by *callback* that a certain rule has fired (described later in section 4.4.5).

4.3.3.5 Composite Event Operators

The set of all events which can be notified within a context is termed the *event domain*. These include events marked on objects, system events, external and default events and named event rules. Composite event operators allow for the monitoring of ordering in event notifications. The following four basic event composition operators represent the minimal operator set needed to express all possible combinations of event occurrences:

- `and (evexpr1, evexpr2)`
  To cause expressions of this form to be detected involves the simultaneous detection of expressions `evexpr1` and `evexpr2`.

- `not (evexpr)`
  Expressions of this form are detected if an event is notified which does not cause the expression `evexpr` to be detected.

- `after (evexpr1, evexpr2)`
  The detection of expressions of this form involves two stages. Firstly an expression of the form `evexpr1` must be detected. From that point, the whole expression is detected on any occurrence of `evexpr2`.

- `every (evexpr)`
  Expressions of this form are detected every time the expression `evexpr` occurs. This will also initialise the enclosed expression to its initial state, for example if `evexpr` is an `after` expression which is detected then its initial condition must recur before it can be detected again.

The primitive `any` which represents any of the events in the event set must also be included. `any` is detected every time another event is notified. Examples of its use are given below.

4.3.3.6 User-Defined Operators

Although any combination of event ordering can be defined using the above operators, other operators can be defined to simplify the expression process. These are specified in terms of the above operators. Examples of such user-defined operators are given below.

- `or (evexpr1, evexpr2)`
  Detected through the occurrence of either event expression `evexpr1` or `evexpr2`. This operator is defined as

  `not( and( not(evexpr1), not(evexpr2)))`

- `prior(evexpr1, evexpr2)`
  Detected if expression `evexpr2` occurs after expression `evexpr1`. The notification of sub-components within the two expressions may overlap. This operator can be defined as

  `and ( after(evexpr1, any), evexpr2)`

- `sequence(evexpr1, evexpr2)`
  Detected on immediately successive occurrences of expressions `evexpr1` and `evexpr2`. This operator can be defined as

  `and( after(evexpr1, not( after(any, any))), evexpr2)`

- `first`
  Fires if and when the first event is notified. Defined as

  `not ( after(any, any))`

- `before(evexpr1)`
  Detected if an event is notified and if at that point in time it is before an occurrence of expression `evexpr1`. This operator can be defined as

  `prior(evexpr1, any)`

- `happened(evexpr1)`
  Detected if an event is notified and it is true that now or at some previous point expression `evexpr1` fired. This operator can be defined as

  `or( evexpr1, prior(evexpr1, any))`

- `before_and_after(evexpr1, evexpr2)`
  Detects every event before an `evexpr1` occurs and then every event including and after `evexpr2`, before the next `evexpr1`. This operator can be defined as

  `every( and( not(evexpr1), not( after(evexpr1, not(evexpr2)))))`

- `frombetween(evexpr1, evexpr2)`
  Detects every event including and after `evexpr1` before the next `evexpr2`. This operator can be defined as

  `not(before_and_after(evexpr1, evexpr2))`

- `any_not_after(evexpr1, evexpr2)`
  Detects every event which is not the first `evexpr2` occurring after the last `evexpr1`. This operator can be defined as

  `not( every( after(evexpr1, evexpr2)))`

- `inbetween(evexpr1, evexpr2)`
  Detected if `evexpr1` is notified after the last occurrence of `evexpr2` without an intervening `evexpr3`. This operator can be defined as

  ```
  after(evexpr1, and( before_and_after(evexpr2, evexpr1),
                      any_not_after(evexpr2, evexpr1)))
  ```

An application developer can construct and use a library of such expressions. They can be specified using a simple grammar.

4.3.3.7 Time Delays

Time delays can also be specified in event expressions using the `delay` operator. An expression of the form

```
delay(<evexpr>,<time>)
```

represents an event expression, the detection of which is delayed by `time` after the notification of `evexpr`. Such an operator can be used to build structures such as *timelines* when certain actions are required to occur at certain times.

The following example illustrates a rule which, if Scarlet is clicked on (i.e. rule `in_position` is fired), 10 seconds later removes the text and restarts `Film`:

```
rule restart = delay(IdentifyScarlet,10.0) -> {
                                                  Info$Display.hide();
                                                  Film$Source.play();
                                                }
```

4.3.4 Manipulating Groups of Objects

Data objects can be grouped into sets using set operators. Available set operators are {`union`, `intersection`, `not`}. Database attributes can also be used in `select` queries to define sets. An example of a set definition is

```
set jkspeak = intersection( select(medium = audio),
                            union(select(subject = Jean),
                                  select(subject = Ken)
                                  )
                          );
```

This specifies a search for all objects in the current context which have attributes `medium` and `subject` with relevant values to indicate they are audio objects of Jean or Ken speaking.

4.3.4.1 Group Invocation

An invocation performed on a set is propagated to every object in the set. The purpose of this is to allow policies to be specified for the manipulation of groups. For example, using the set defined above, an invocation of the following form could be used to `stop` any set members which are playing:

```
jkspeak$Source.stop();
```

Thus groups of objects can be started, stopped and manipulated using one invocation. One emerging use for this is in policies for QoS. In some situations applications may be notified that they must degrade usage of network resources. Alternately an application may require a new video connection but has not been allocated enough network, device or processing resources to make this possible. In these situations it can decide to degrade all its existing video connections or turn off all its audio objects, to enable it to continue object presentation or allow the new connection to start. Policies can be encoded of the form "degrade all source video objects to 50% and cut the frame rate to 10%" and if that fails "turn off all audio". To realise this would require group invocations on sets of video or audio objects.

4.3.4.2 Synchronising the Presentation of Data Objects

The `sync` declaration can be used to specify that certain objects are to be synchronised in their playback. An example use of this facility is in *lip-sync'ing* between video and audio. The term lip-sync' has derived from the synchronisation of lip movements to words being spoken audibly. A set of objects must first be defined. This set must then be specified to be synchronised. An example which would synchronise the playback of the `Film` video object and the `Soundtrack` audio object is shown below.

```
set Movie = union(Film,Soundtrack);
sync Movie;
 ....
 ....
Movie$Source.play();
```

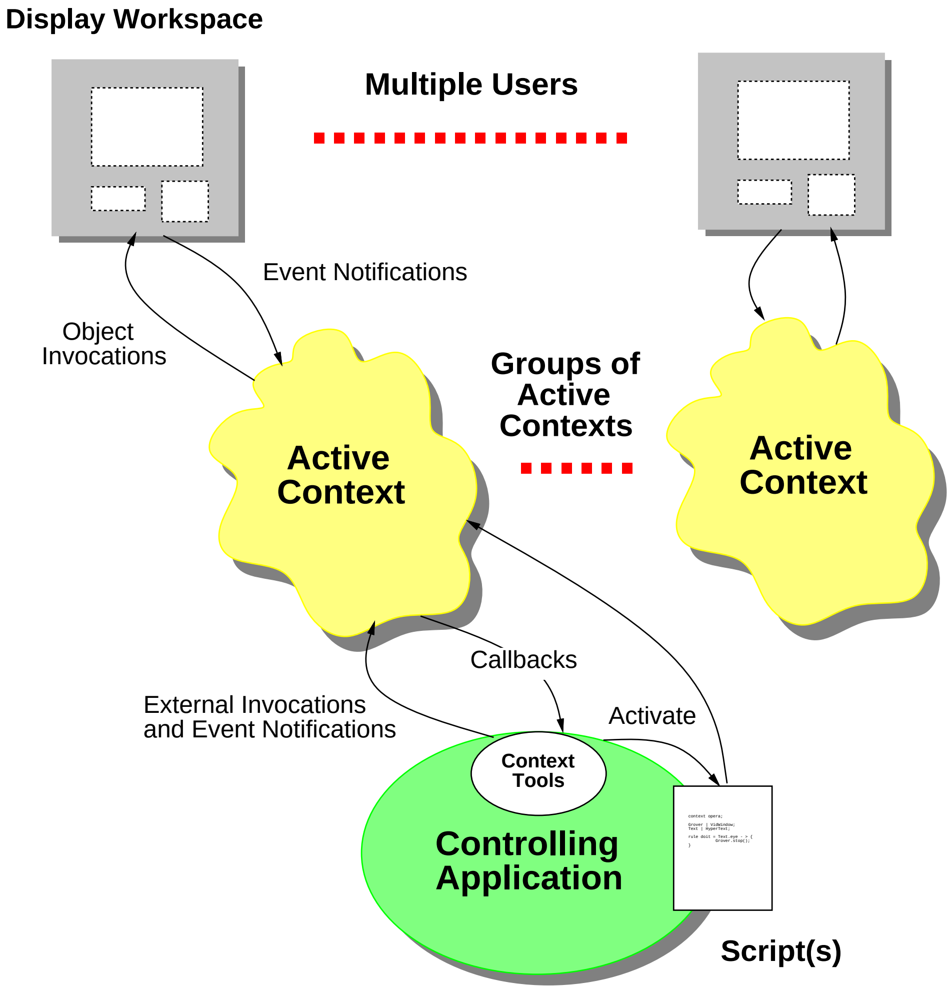


Figure 4.8: The controlling application

4.4 Application Management of Presentation

An overriding requirement of this authoring model is that it must not represent a *closed system*. It must be possible to make interactive presentations *part* of a developer's application.

Another requirement is brought about by the fact that a cooperative application may involve many users. This implies many instances of the same script all of which are cooperating in tasks. Therefore it must be possible to manage multiple cooperating presentations.

When a script is realised by applying it to a presentation database context, an *active context* is created to manage interactive presentation. *Context tools* are provided so that a controlling application can activate, communicate with and control such active contexts. These tools are usually provided in the form of a library to be included by an application. Figure 4.8 illustrates the relationship between the controlling application and active contexts managing interactive presentation. This section describes the required facilities of the context tools interface and illustrates with examples from the prototype interface (written in the C language).

4.4.1 Activating Contexts

Scripts are written separately from the controlling application. Each script has a name which is used to reference it in the creation of an active context. The operation below is used, here shown activating a script called `Lab.imp`.

```
ContextIR = ActivateContext(``Lab.imp'', < Instantiation Parameters >);
```

The `<Instantiation parameters>` are actual values used to instantiate a context template. For the example script from section 4.3.1 the instantiation parameters would be the names of audio and video devices to be used by the two callers.

This process takes the script, parses it and realises it as an active context. If this process is unsuccessful an appropriate error message is reported. If it is successful then an interface reference for the context is returned to the application. This is used in all subsequent contact with the active context.

4.4.2 Displaying a Script Workspace

To display the *workspace* in which the presentation described in a script will be presented (see figure 4.5) requires the operation

```
AContextRealise(ContextIR,
                <WorkstationDisplay>,
                <ApplicationName>,
                width, height);
```

The `WorkstationDisplay` parameter specifies a workstation screen on which the workspace is to be created. `ApplicationName` specifies what this presentation instance is to be called and `width` and `height` describe the dimensions of the workspace. All display components created in a script are presented in its workspace.

4.4.3 External Events

As well as the IMP language being able to handle named events which have been marked on objects in its context, it can also handle named events which are signalled from outside the system. The `external` script statement (see section 4.3.3.3) allows for this. One source of such events is the controlling application. Returning to the Laboratory presentation example, consider how a script would start. It must receive an event from somewhere to initiate presentation. Within the script, declarations such as those shown below could be defined.

```
external StartMovie;
....
....
rule Go = StartMovie -> {
                          Film$Display.show(50,100,200,200);
                          Movie$Source.play();
                        }
```

This allows for an event `StartMovie` to be notified externally, i.e. not from one of the objects in the context. If such an event is notified then rule `Go` is fired and the video starts playing.

Within the controlling application an operation of the form shown below can be used to notify scripts of event occurrences.

```
AContextExternalEvent(ContextIR , <Event Name>)
```

In the case of the script example above, an external event operation using the event name `StartMovie` would cause the `Go` rule to fire. The presentation will thus commence since other event rules will be fired from event occurrences associated with objects `Film` or `Soundtrack`.

4.4.4 External Invocations

In some applications, only allowing invocations on objects to be specified in a script in response to event notifications can be restrictive. The IMP language does not provide any logical or arithmetic functions thus the parameters of method invocations in a script are fixed. The script language is designed to be a minimal language which can specify interactive presentation and not a general purpose language. Some applications require dynamic operations such as on-the-fly re-layout of display components.

It is thus often desirable to alter method parameters dynamically. It is possible to use the script to specify event monitoring requirements and instruct it to notify a controlling application of the firing of specific event rules (described further in section 4.4.5). Facilities are provided for a controlling application to perform invocations itself. Operations of the form shown below can be performed by a controlling application.

```
AContextExternalInvocation(ContextIR,
                           <Object Configuration>,
                           <Presentation Object>,
                           <Method>,
                           <Parameters>);
```

4.4.5 Application Callbacks

The firing of event rules can be notified outside of a script through application callbacks. A controlling application can specify a routine to be called if a specific rule fires. Within such a routine, other active contexts can be contacted or new presentations started. Callback routines are declared with the following header:

```
void Callback_Handler(Context, RuleName, Clientdata)
```

Active contexts are informed of a callback using an instruction of the form

```
AContextAddCallback(ContextIR,
                    <RuleName>,
                    <CallbackHandler>,
                    <ClientData>);
```

The `<ClientData>` parameter is a pointer to a memory structure which is returned to a user in the `<CallbackHandler>` routine.

4.4.6 Context Groups

Extending the notion of invoking methods from the controlling application, facilities for invoking methods in groups of active contexts are provided. It is possible that these contexts are being presented on different workstations. This provides a facility for the management of cooperating instances of the same presentation from a single centralised control. For example within an audiovisual conference application, the addition of a new conferencee may be required. The new user must appear on every participant's display. Rather than performing this operation on every active context individually it is useful to use one operation only.

The facilities of external event notification, external invocation and handling callbacks are available for both single or groups of contexts. Context groups can be created using the operation shown below which returns an interface reference with which the group can be controlled.

```
GroupIR = AllocGroup();
```

The interface references of individual contexts can be added to groups with the operation

```
AddToGroup(GroupIR,ContextIR);
```

Contexts can also be removed from groups with the `RemoveFromGroup` instruction. The methods to control group contexts are `GroupAddCallback`, `GroupExternalEvent` and `GroupExternalInvocation`.

A collaborative application can be implemented as a shared service. Users joining an application session invoke the appropriate service operation, a context is activated for them and they are added to the context group. Alternatively, applications can start up separately and at some point agree to join their sessions together. In this case one may be nominated as *master* and given the active context interface references of the other, to be added into its context group.

4.5 Range of Support

4.5.1 Supported Application Types

Using this authoring approach it is possible to support a wide range of applications. The following observations can be made with reference to the analysis in sections 2.1 and 2.2:

- Data objects and configurations of presentation objects support the construction of presentational applications (described in section 2.1.1). Presentation objects take into account application requirements in the form of presentation methods and system requirements in the form of quality methods. They can support functions such as device access, analysis, processing and display. Objects can be grouped together for group invocation or synchronisation.
- Event-driven rules support the construction of compositional applications (described in section 2.1.2). Events marked on data objects can be handled and actions taken. Using this approach structures such as timelines and hypermedia links can be supported.
- Synchronous collaborative applications (described in sections 2.1.4, 2.1.5 and 2.1.6) can be supported using a controlling application acting on event feedback from multiple active contexts. Asynchronous collaborative applications (see section 2.1.3) could transmit data items along with modelled data and script information in a marshalled format. A program to unmarshal them at the receiving side is also required.

In chapter 7 prototype applications built using the IMP platform will be used to illustrate these assertions.

4.5.2 Tool Support

Tools to assist in application development can be built over the authoring model. For example, Graphical User Interface (GUI) builders [**Cardelli88**] allow display objects to be placed and sized using direct manipulation. Different types of tool may be appropriate for different types of application. For instance, techniques to simplify the development of conferencing applications are different from those appropriate to the development of hypermedia applications.

It is also possible to develop tools to map other authoring models onto the IMP approach. Some of the models illustrated in chapter 3 could be supported such as [**Fujikawa91**, **Vazirgiannis93**] or the graphical approach to configuration in [**de Mey92**].

4.5.3 System Extensibility

It is important that as systems expand and software components are developed, such changes can be supported by the IMP platform. New medium types, event types and presentation object classes can be added to the system. The way in which new support can be added is described further in chapter 5. New facilities can be used through the same authoring system. For example, addition of a new presentation object type allows it to be referenced in a script in the same way as any other.

4.6 Conclusions

An approach to the authoring of interactive presentation for multimedia applications has been described. It provides construction facilities which support a wide variety of applications. These facilities are easy to use, employ reuse and tailorability, and yield portable and maintainable applications.

Data objects can be selected from a database of components. Indexing facilities enable relevant objects to be retrieved using queries. Data items not referenced in the database can be added easily regardless of source device, medium or format. Data objects can be tailored to serve a particular application, by annotating attributes and marking events. Related instances can be grouped together into a context.

A script can be used to specify the interactive presentation of objects in this context. Configurations of presentation objects can be specified. These provide user-defined combinations of analysis, monitoring, processing and display functions on data objects. The notification of events associated with data objects can be used to invoke methods on presentation objects and thus influence the presentation process. Particular patterns of event occurrences can be monitored for and also used for this purpose. Data objects can be grouped together as sets for group invocation. Synchronisation of objects for functions such as lip-sync'ing can also be specified.

A controlling application can activate scripts as active contexts. These manage the interactive presentation specified in a script. They can also communicate with the application at run-time. Groups of collaborating active contexts may be defined, and communicated with as one.

Chapter 5

Object Support

The previous chapter introduced the use of objects as the building blocks for modelling and presenting multimedia data. The *object support* layers of the presentation platform provide the facilities for the realisation of these objects. This chapter describes the object support facilities first in general terms and then using a specific example of a complete implementation developed within the IMP prototype platform.

5.1 The Role of Object Support

Object support implements an object-based abstraction of a multimedia environment. This has the following two aspects:

- *Implementing data modelling features*
  As described in section 4.2, the data modelling interface can be used to create data objects which reference distributed data items. These objects can be tailored with attributes and events, to represent their run-time characteristics. Related presentation data can be grouped together in contexts. All these functions must be provided. Facilities for type-checking data modelling operations are needed. Also, an underlying storage scheme for modelled data is required.

- *Providing facilities for creation, configuration and control of presentation objects*
  Presentation objects (as described in section 4.3.2.1) perform a run-time presentation function on data of a certain medium type. They can be controlled using their methods. A scheme for implementing these objects is required. Facilities must be provided for their dynamic creation and to allow them to access the data referenced by data objects. Also needed is the means to connect them together in legal configurations, to enable them to manipulate the same data.

Presentation services (described later in chapter 6) use object support to realise authored presentations. Object support is separate from the presentation service layer in that the implementation of its components are dependent on a particular environment.

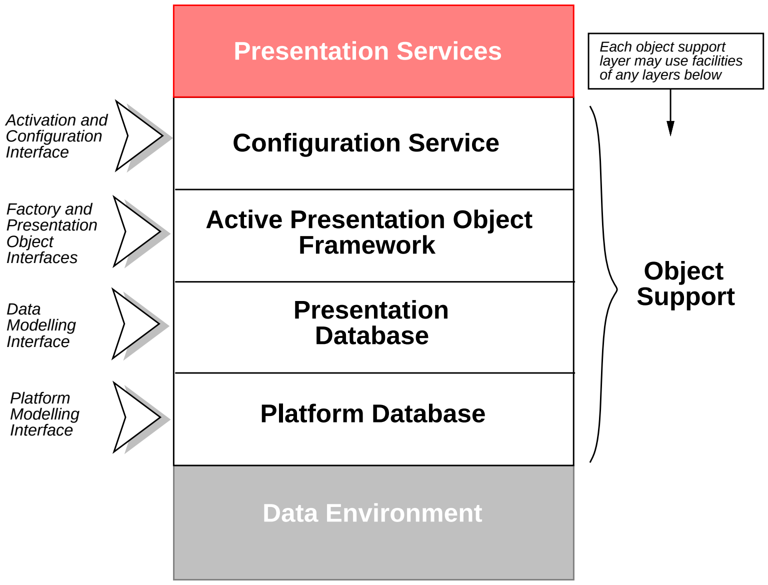


Figure 5.1: Object support

5.2 Object Support Design

To implement these roles requires both a storage scheme for presentation data and a novel architecture for run-time presentation. A decomposition of the proposed object support design, within the presentation platform, is illustrated in figure 5.1. Each layer is able to use the facilities of all the layers below it. The object support layers are

- *The platform database*
  This stores information on the data environment and platform infra-structure which operates on it. It provides an interface through which information can be updated or queried. It is described in detail in section 5.3.

- *The presentation database*
  Using information stored in the platform database, data objects can be realised. These are implemented as persistent objects with methods to manipulate their data. Issues pertaining to the design of a presentation database are discussed in section 5.4.

- *The active presentation object framework*
  Active objects are used to realise the functionality of presentation objects. They have data, methods and autonomous threads of control. Through their methods they can control the nature and quality of presentation, be informed of events to monitor for and be connected to other compatible objects. Creation of these objects is managed by factories. This framework is described in detail in section 5.5.

- *The configuration service*
  The configuration service, described in detail in section 5.6, can activate and connect presentation objects appropriate to a particular data object.

This chapter describes a detailed design of the object support layers. Section 5.7 outlines how the object support design was realised in the prototype platform.

5.3 The Platform Database

It is necessary to provide facilities for storage and manipulation of information regarding the support offered by the platform. This information is likely to vary because

- Object support facilities differ between environments due to differences in the available devices and supported formats.
- It is possible to adjust the object support facilities within a platform (as described in section 4.5.3). For example, it is possible to add support for a new medium, format, device or event type.

This functionality is provided by the platform database. All levels within the platform can use this service to validate operations. This section describes what information must be managed and how it can be changed.

5.3.1 Required Platform Information

The platform database can store *types, templates* and *relationships*. Types represent supported features such as medium types and format types. Templates are types which have to be supplied with further information to be instantiated. Relationships represent associations between types and templates. To instantiate some objects, such as data objects, requires several types and templates which must be related in a *dependency* relationship.

Example types of information which must be stored by the platform database are

- *Supported medium types*
  The database must be informed of all medium types support such as text, pictures, video, audio and audiovideo.
- *Supported format types*
  The database must also store data indicating the support implemented for different formats of each medium type. For example, for video, formats Pandora, JPEG and MPEG could be supported.
- *Device templates*
  Also recorded is information about what devices are supported by the platform and what information must be provided to reference data which originates there. For example, for a MSSA data item, a storage service identifier (SSID), which uniquely identifies any object, is required; whereas for NFS data, a domain pathname is required.
- *Event templates*
  The database stores which types of event exist in the system. For each event type a template of information must be stored specifying the parameters of the event.

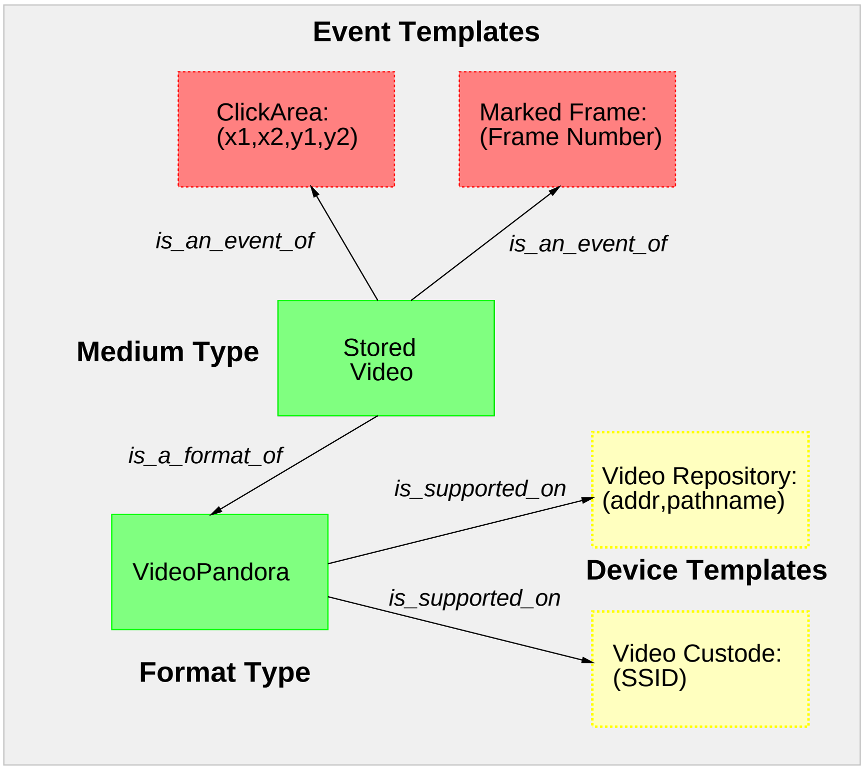


Figure 5.2: Example data object dependency

- *Data object dependencies*
  Within the database are stored dependency relationships which specify which types of medium data can originate on which devices and support which formats. Which event types are applicable to which medium types must also be recorded. Figure 5.2 illustrates part of a data object dependency for stored video.

- *Presentation object class hierarchy*
  Knowledge of the class hierarchy of presentation objects (introduced in section 4.3.2.1 and continued in section 5.5.1) is also recorded. For each class, information is stored such as the name of the class, its methods, its sub-classes, the medium type it processes, its input and output formats, the device it accesses and information required to locate the data on that device.

Other information is emerging as desirable to store in this manner (see section 8.2.2).

5.3.2 Platform Database Interface

The platform database has an interface which allows the above information to be manipulated. For each of the above classes of information it must be possible to

- *Add new information*
  As new support is added to the platform, the database must be informed. For example,

adding support for a new medium type involves informing the database of formats supported, event types supported and details of presentation objects which manipulate it.

- *Query existing information*
  Methods must exist to query and retrieve database information. Example queries are listing all supported medium types, listing all event types which can be marked on objects of a certain medium type or listing the methods provided by a particular class of presentation object.

- *Remove existing information*
  This is required if supported services are decommissioned.

Information stored in the platform database is used at every level within the object support framework and by other higher levels. This use will be illustrated in the following sections. Tools can be built to assist a platform maintainer in keeping a record of platform facilities.

5.4 The Presentation Database

The presentation database realises the functionality of the data modelling interface described in section 4.2. The internal structure of a presentation database and example operations are illustrated in figure 5.3.

5.4.1 User and System Methods

The aim of masking unnecessary heterogeneity means that application builders can manipulate database objects without knowledge of their system-dependent details (as described in section 4.2). However, database components must provide methods for system components (such as the configuration service, described in section 5.6) to be able to access these details.

5.4.2 Instantiation and Type Checking

Instantiation operations require type-checking with the platform database to validate their semantics.

Object creation requires a name, class and data reference. The class represents a valid dependency between medium, format and source device types. In order to ensure that such a dependency exists, the platform database must be consulted. For example, figure 5.2 shows that a valid data object type can have medium type stored video, format type Pandora and device type video repository or video custode. The data required to create the object is the information required to instantiate the relevant device template. For example, to create an object of type `Video_VideoPandora_VideoCustode` requires an SSID referencing the video data on that custode.

When marking events, a name for the event, a type and relevant data must be supplied. In order to ensure events are added correctly, event information from the platform database

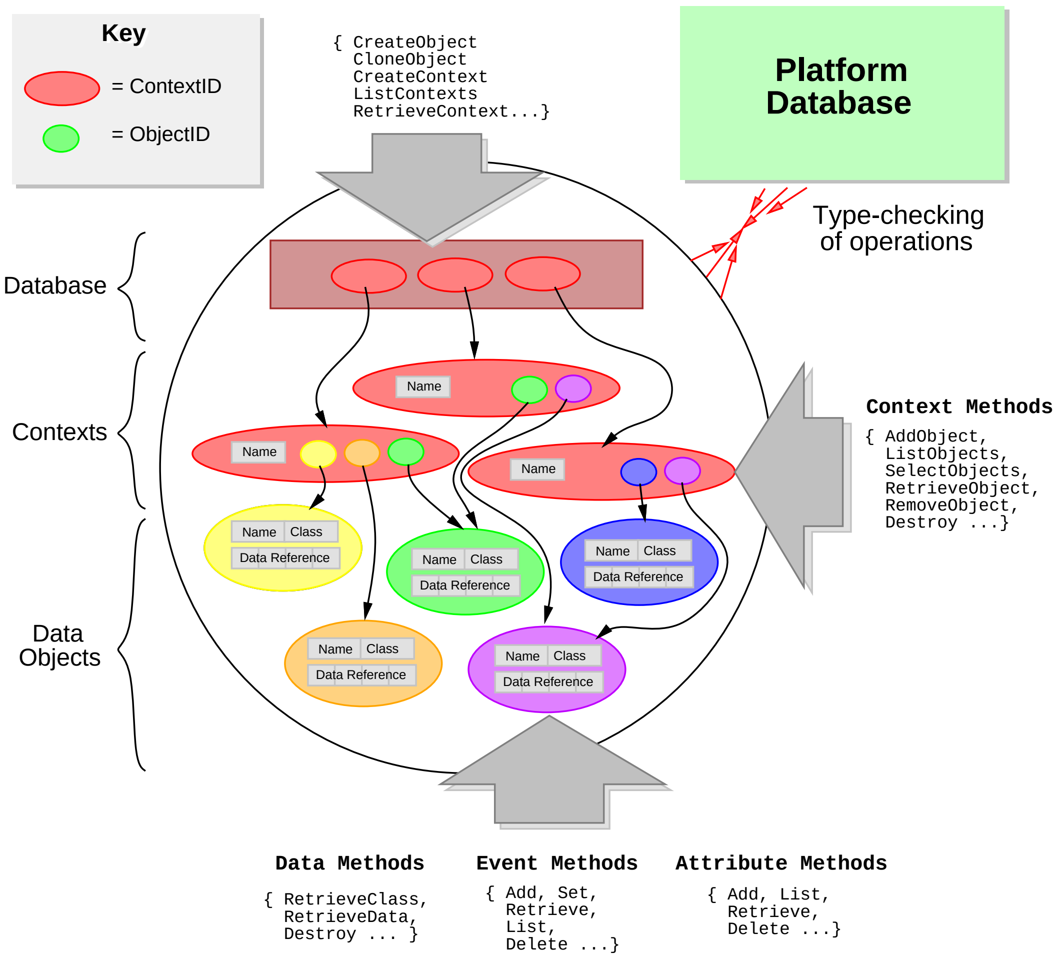


Figure 5.3: Database organisation and methods

can be queried. As illustrated in figure 5.2, some valid event types for marking on video objects are `MarkedFrame` and `ClickArea`. The data associated with the marked event is the information required to instantiate the event template. For example, for `ClickArea` events it is four coordinates and for `MarkedFrame` it is a frame number.

In this model, attributes and contexts are currently untyped. As no semantic information is associated with them, no platform database look-ups need be performed when adding them. However, typing of either of these can be envisaged (see section 8.2).

5.4.3 Names and IDs

A distinction is necessary between names and IDs. A name is a user-friendly identifier, unique within a certain scope, e.g. an event name is unique within an object. An ID is an opaque type, unique throughout the database. Each object which can exhibit autonomy from its parent should be allocated an ID when created. Creating contexts and objects returns `ContextID`s and `ObjectID`s respectively. A context is stored as a sequence of `ObjectID`s. Through this mechanism, objects can be shared between contexts.

An ID is required to access object methods. Contexts offer composite object query and retrieval operations which perform an operation on all contained objects. For example, a `SelectObject` method retrieves object attributes and searches for particular values.

5.4.4 Physical Storage of Data

The presentation database is concerned with persistent storage of

- *Presentation meta-data* - i.e. names, classes and instantiated templates.
- *Object references* - i.e. `ObjectID`s and `ContextID`s.

Possible techniques for storing this information are

- *Structured files*
  Services which provide type constructors to define storage structures have been built. These can be used from within programming languages to store and retrieve data. An example system, the structured file custode, is described later in section 5.7.5.2.
- *Persistent languages*
  A persistent language allows types within a programming language to *persist*. This means the data associated with variables defined from these types is retained when the program is not running.
- *Object-oriented database systems*

The level of assistance offered to database implementors is subsumed and increased by each item on the list above. Structured file systems are likely to offer constructs for building fine-grained concurrency control facilities. Persistent languages may offer transactions mechanisms [**Wu93**]. Database systems offer automated query searching [**Korth91**]. Any required features not provided by the underlying storage system must be implemented from scratch. An appropriate prototype implementation is illustrated later in section 5.7.5.

5.5 The Active Object Framework

This section describes how the system dependent aspects of presentation objects are realised using active objects.

5.5.1 Realising Presentation Objects

At the authoring level, presentation objects represent functions which are applied to particular medium types. Within the object support levels, the system specific aspects of this functionality must be provided. For example consider the video object `Film` from the authoring chapter. It can be manipulated by video presentation objects. In order to remove it

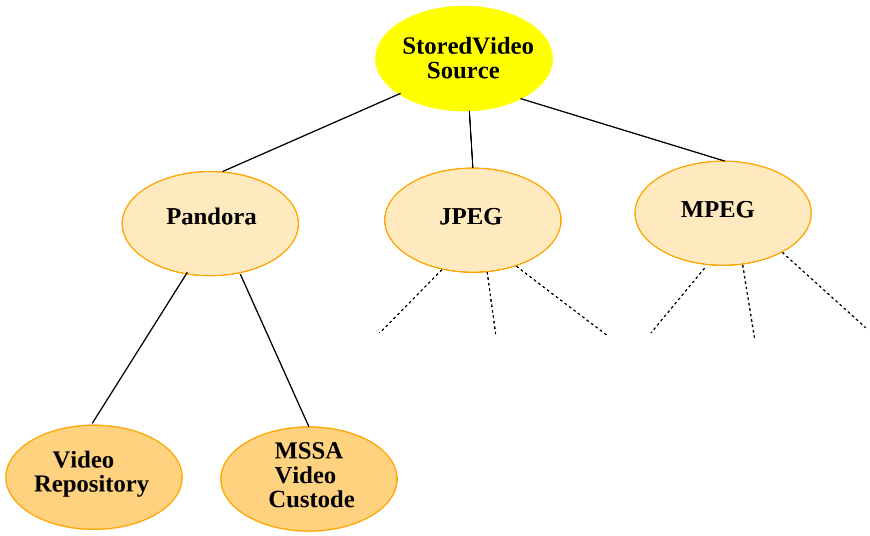


Figure 5.4: Continuation of a class hierarchy

from its source device a source video object is created with methods {`play ..... stop`}. In reality, many types of video may be supported by the system such as JPEG, MPEG and Pandora. These types can be supported on many types of device such as the video repository or the MSSA video custode. For each of the possible format and device combinations, sub-classes of the medium presentation class must be provided. These must have the same methods as the parent so differences are transparent to the application developer. As an example, figure 5.4 shows how the class hierarchy presented in figure 4.4 can be extended to support environment heterogeneity. Each leaf node is implemented as an active object class (described below). Assuming the object `Film` is of format Pandora and stored on the video repository, a source object of class `Source_Video_Stored_Pandora_VideoRepository` would be required. Details of available types of component and the formats which they can process are stored in the platform database.

5.5.2 Active Presentation Objects

Active objects not only have private data and methods but also are instantiated with threads of control [**Kim89**, **Blair91**]. This means that, as well as performing operations on receipt of a message, they can also perform asynchronous processing and send messages. Many such objects can run concurrently.

The autonomous concerns of active presentation objects are

- *Presentation functions*
  This involves taking in data from a device or another presentation object, performing some function on it and outputting it, or storing it on a device. This process is described in section 5.5.5.

- *Event monitoring*
  This involves checking whether events associated with presentation of the data have occurred. This process is described in section 5.5.6.2.

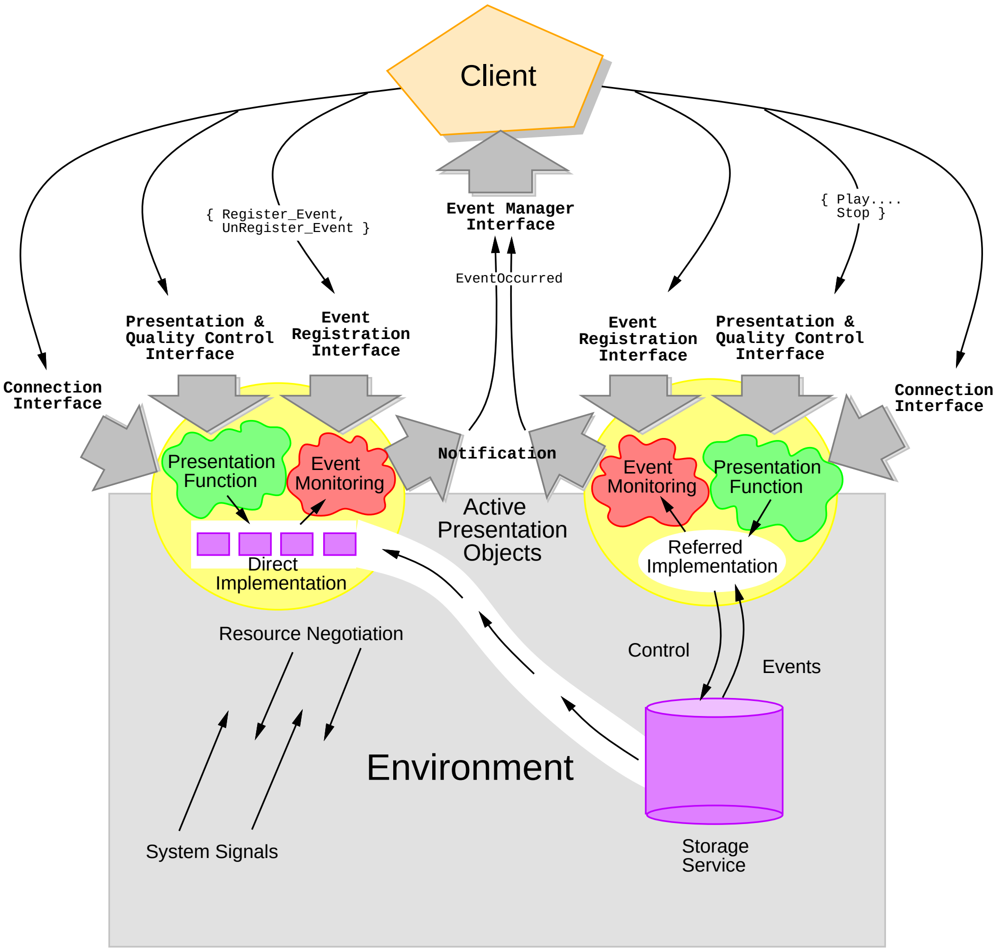


Figure 5.5: Active Presentation Objects

- *Handling invocations*
  By invoking methods, clients can initialise and influence at run-time the presentation and event monitoring processes. The interface of an active presentation object is described in section 5.5.3.

- *Notification*
  If events are detected at run-time then they must be notified to relevant clients. This process is described in section 5.5.6.3.

Each of these duties must be performed in a timely manner. If threads are not preemptable then they must perform their task and block so that others can take over.

5.5.3 The Interface to Active Objects

Active presentation objects have methods for the following purposes:

- *Presentation control*
  Methods to affect presentation functions, e.g. for a source object {`play, ......` `stop`} or for a display object {`show`, `hide`, `move`, `resize .......`}.

- *Quality control*
  Methods for adjusting resource utilisation, e.g. for a video source object {`set_resolution`, `set_framerate ....`}. Methods with such functionality generally only apply to continuous media objects.

- *Interconnection*
  These methods are used to connect presentation objects together in configurations.

- *Event registration*
  These methods are used to tell a presentation object of events to monitor for. For example, if a synchronisation object is able to monitor for video frame boundaries, it must be told which ones are of interest.

Only the first two categories of method are accessible from the script language. The others are used by the system to pre-configure the presentation objects automatically.

5.5.4 Development Techniques

This section discusses factors which influence the way in which active presentation objects are implemented.

5.5.4.1 Defining the Interface

When a new presentation object class is devised, an interface must be defined for it. The same interface must be provided regardless of the implementations for different format and device types.

The selection of methods for the interface must be based on the features perceived to be necessary, currently and in the future, to control objects of such a presentation class.

It is likely that some implementations of a presentation object will not be able to realise some of these methods as the underlying system cannot support them. It is also likely that, as new systems with increased functionality are built, unenvisaged requirements will emerge. In such a situation, the methods for the presentation class can be updated or a new class can be devised.

5.5.4.2 The Influence of Existing Infra-structure

There are two main techniques which can be used interchangeably in the realisation of active presentation objects.

1. *Direct implementation*
   Implementing the functionality from scratch. Active objects must manage the input of data, any processing involved, monitoring for events and outputting of the data.

2. *Referred implementation*
   Implementing on top of existing infra-structure. Within an environment which already has device control libraries and application services which can retrieve, manipulate and display data there is often no need to implement some components from the bottom up. The active object interface can be implemented and driver routines used to control relevant system components in response to invocations.

A combination of these techniques can be used. For example, it may be possible to use existing infra-structure for access to devices or display, but necessary to develop parts of objects from scratch such as routines to monitor for specific types of event.

The first technique encourages new software to be developed within the platform. For example, encapsulating a newly developed algorithm to process video as an active presentation object, as opposed to a stand-alone program, enables it to work with other compatible presentation objects and contributes to reusability and tailorability.

The second technique allows software which is already in existence to be incorporated within the platform.

5.5.4.3 The Influence of Medium Type

The nature of the medium being manipulated affects the realisation of an active presentation object. Different techniques are required for

- *Still media*
  Still media components do not require continuous presentation. Once data has been accessed and presented it exists in a static form, e.g. text or a picture. No presentation thread is required to perform any actions on it, although continuous event monitoring threads may be required. Data is only affected through the invocation of presentation control methods. No method of quality control is required.

- *Continuous media*
  Continuous media requires continuous presentation and event monitoring.

Exceptions to these rule are active objects which transform still media into continuous media. Examples are live text or live picture objects. These can be used in conferencing applications to represent text communication channels and shared whiteboard facilities. Some display components can transform static media into dynamic, e.g. a scrolling text display object.

5.5.5 Presentation Functions

This section describes implementation approaches to the presentation functions performed by active objects. The initialising and controlling of presentation functions involve

- *Presentation control*
  The presentation algorithm and associated presentation control methods which can affect this process.

- *Connection control*
  The means to connect together compatible presentation objects, controlled through the connection methods.

- *Quality control*
  The ability to alter the processing and transport resource usage of presentation objects through the quality control methods.

5.5.5.1 Presentation Control

Depending on the object in question, presentation functions may take the form of device access, analysis, processing or display of multimedia data. Implementing such presentation functions directly, requires additional threads of control to manage data input, output and the presentation function performed on it. For continuous media, this process must be performed on a continuous time-regulated basis for every sub-element. An example is an object which processes video images by inputting frames into a buffer, performing the processing algorithm on them and outputting them. Presentation control methods are used to alter parameters of the processing algorithm.

Alternately this process can be managed by an existing part of an environment. Using such *referred presentation*, object methods cause control instructions to be sent to the environment component managing the presentation function. It may be necessary to have a separate thread to handle presentation-related callbacks from such a component. An example is a stored audio source object which sends instructions to an audio storage service to make it perform operations on a particular audio file, such as play, stop, fast wind or slow wind.

5.5.5.2 Connection Control

All object classes have methods to interconnect to other compatible objects. If an object is built from scratch then *direct connection* must be implemented. If possible, connection techniques should be compatible among related objects. Methods employing shared memory can be used. This involves passing the address of memory areas containing the presentation data to the object which requires the data next. The organisation of memory must be known mutually amongst objects. This method is efficient in that it saves copying large amounts of data between address spaces (which is expensive) but it does force objects to exist within the same address space. Another method is connection using stream endpoint abstractions such as ports. Objects are passed a port reference which they can connect to using appropriate protocols. Objects can thus exist in separate address spaces. Depending on environment and object class, objects may be required to negotiate an end-to-end QoS for a connection (see section 5.5.5.3).

Referred connection can often be used for objects which are implemented on top of existing infra-structure, as connection may already be provided as part of the service being used. In this case objects must just send messages to the service, requesting the required connection. For example, it is likely that the audio storage service (introduced in the last section) will have been implemented with methods to send data to a specific client. Thus the stored audio source object need only specify this client, rather than managing the entire process.

It may be desirable to connect existing object classes which are incompatible in some respect, e.g. they process a different format type or have different connection mechanisms. In some cases it is more efficient to implement a *converter* object to interface the two than to implement new compatible classes. For instance, with reference to the example problems, in the first case the format must be converted from one type to the other; in the second case a converter must connect to both and interface their data transfer. A library of such converter objects can be built up and reused (see section 5.6.5).

Despite the authoring interface only allowing objects to be connected in *pipelined* configurations, it is possible to implement objects which have multiple inputs and outputs [**de Mey92**, **Coulson93**]. Examples are *mixing components* which mix aspects of multiple audio objects or *display layout components* which take in variable numbers of video objects and manage their layout, as in a conferencing application.

5.5.5.3 Quality Control

QoS issues can affect environment components including source devices, networks and operating systems. A source device such as a storage service has limited amounts of bandwidth to its disks; networks have a limited bandwidth for data transport; operating systems have a finite amount of processing resources.

In such an environment, adequate resources for presentation and an end-to-end QoS for a connection are required. To achieve this, a presentation object has to negotiate a level of QoS with all environment components involved, for a session. System signals associated with QoS, such as an environment component indicating the need to renegotiate a level of QoS, are notified from a presentation object in the same way as it notifies detected user events.

The quality control methods are provided so that the resource usage of a presentation object can be adjusted. A source object allows adjustment of factors such as quality or rate in data being drawn from a device. Source objects which control live devices such as cameras or microphones can instruct the device controller to digitise at a lower quality. Objects which control access to storage services are more problematic but it may be possible to inform the storage service to skip frames. Otherwise a separate object on the same host can degrade the data quality before transmission. Objects connected to a source object are affected by such operations. These objects are either able to detect changes in data rates or can be informed of such changes by the object *up-stream* from them. This is possible as objects can be passed the interface reference of objects connected to them, during configuration (see section 5.6.4). Other classes of object can have quality control methods. Synchronisation objects, for example, can have methods to specify tolerance to jitter. This affects the amount of memory they allocate to buffering.

Each presentation object has a default quality setting. Relevant system events are notified if an object cannot acquire adequate resources for its processing and transport requirements or is denied them at some point. It is up to a client to use these events to adjust resource usage.

5.5.6 Event Functions

This section describes the event functions performed by active objects. Initialising and controlling these involves

- *Event registration*
  Informing an object of events to monitor for.

- *Event monitoring*
  The particular algorithms used to detect occurrences of these events.

- *Event notification*
  Informing interested clients if these events occur.

5.5.6.1 Event Registration

All active objects inherit an event registration interface which allows them to be informed of events they must monitor for. Parameters are passed to indicate the name, type and parameters of the event to register, for example `MarkedFrame` and an associated frame number. A *callback reference* to an interface on which the active object will notify that the event has occurred, is also passed.

Active objects can only monitor for events which they have been programmed for. Information regarding which classes of active objects monitor for which events is stored in the platform database. It is thus possible for a client to check whether an event registration invocation is valid before initiating it. Active objects add registered event information to internal data structures called *event monitoring tables.* These are structured in different ways depending on event type (examples of which are given in section 5.7.6.4).

5.5.6.2 Event Monitoring

Event monitoring techniques are different dependent on the type of event being monitored for and the type of active object performing the monitoring.

Direct event monitoring can be used by objects which implement their own presentation functions. The presentation thread is used for event monitoring, for example within a video presentation object every video frame which is processed can also be checked to see if it is associated with an event occurrence.

Referred event monitoring involves an active object instructing an underlying service to monitor for particular occurrences. It can be informed that an event has occurred or that a check should be made to see if an event has occurred. For example, a windowing system can inform a display object of event occurrences associated with its user interface.

5.5.6.3 Event Notification

If events are detected during event monitoring then they must be notified to the appropriate client. To manage event notification requires another thread of control. When scheduled, the notification thread informs the relevant client of event occurrences.

As already stated, when an event is registered, a callback reference is passed. Clients are notified on this interface of event occurrences and event details are passed as parameters. Clients must manage the creation of objects to handle the callbacks. A client requires multiple threads of control in order to invoke methods and handle incoming notifications concurrently.

5.5.7 Active Object Manufacture

This section describes the services a client must use to request the creation of particular active objects at run-time.

5.5.7.1 Factories

A request to create an active object of a particular type must be referred to a *factory* object. Traditionally, factories (such as in the ANSA model [**ANSA90**]) manage the creation of objects of a particular type. In this model, factories can manage the creation of a particular range of sub-classes. Examples are factory objects that manage all synchronisation objects or all display objects.

Factories and the objects they create exist in *factory sites*. A factory site must exist within a process to provide the threads of its constituents with processing resources. Factory sites can run in separate processes from one another and on separate machines. Obviously objects which must run within the same address space must either be created by the same factory or their factories must run within the same address space.

Factories provide creation methods which take a presentation object class as a parameter and return the interface reference of the created object. The interface reference can later be used to destroy the object. When creating the object, a data pointer is supplied which contains the class-specific data required by the object. For example, the `Film` object which accesses Pandora format video from a video repository would require the address of the relevant repository and a repository pathname (see figure 5.2). A callback reference must also be supplied. This is used by the object to notify any system events, such as an inability to negotiate sufficient resources.

The management duties of factories is entirely up to the platform administrator. All objects may be created by the same factory or each class may have its own. Policies for factory site placement are usually determined by load balancing requirements and objects' abilities to exist in separate address spaces.

5.5.7.2 Factory Builders

In some cases dynamic instantiation of factories is required. An example is the case of display factories. A display factory is unusual in that when it is created it manufactures an associated workspace. As described, a workspace is a parent window into which display objects can be laid out. Any display objects created by a display factory can be viewed in the associated workspace. For each application, one or more new workspaces and thus display factories are required.

Instantiation of factories is managed by *factory builders*. Factory builders run within a factory site and any factories created exist in that site.

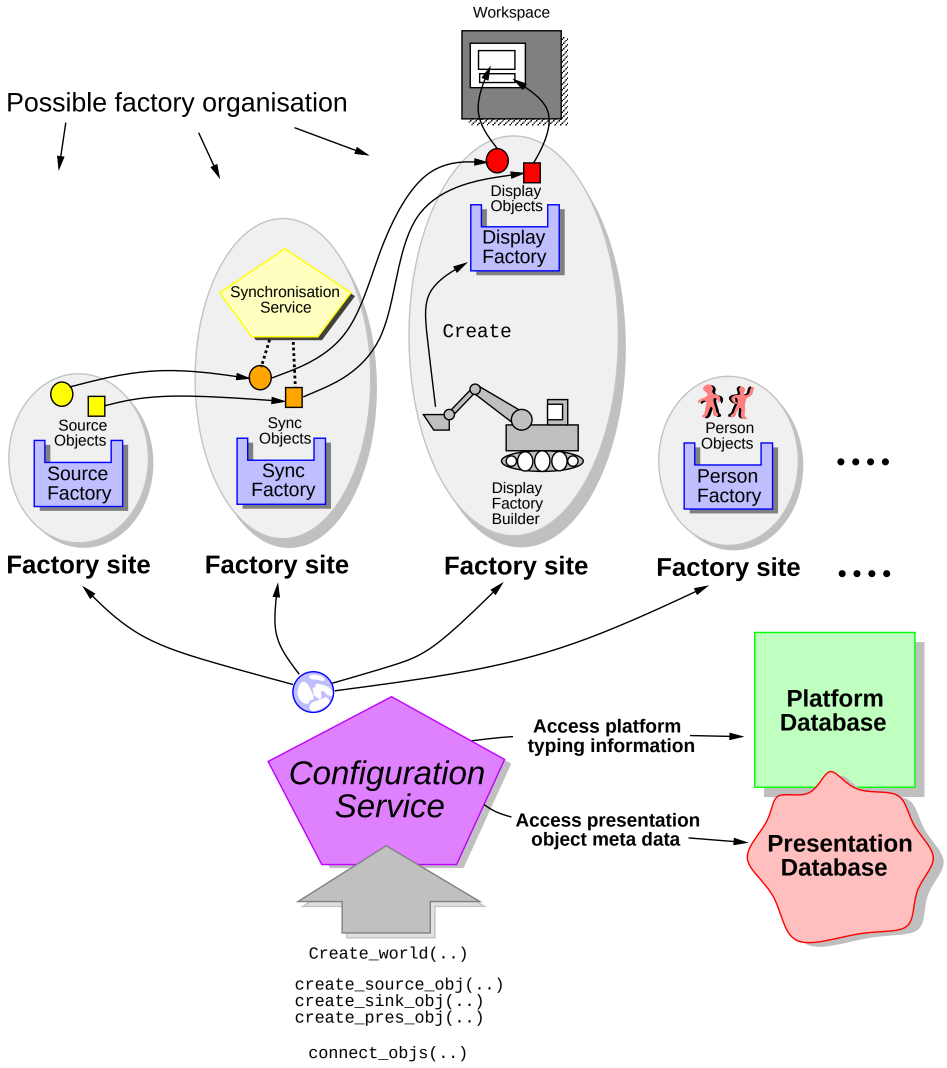


Figure 5.6: Factories and the Configuration Service

5.6 The Configuration Service

The configuration service provides an interface through which clients can activate and configure presentation objects, with reference to a specific data object, without knowing their environment-dependent details. Figure 5.6 shows the operation of the configuration service with relation to the other object support layers. This section describes its main functions.

5.6.1 World Creation

When factories and factory builders are started up, their interface references are registered with the configuration service, and it is informed what object classes they can create. In order for a client to create an active presentation object, a reference to the relevant factory must be obtained. At the start of an interactive presentation session, references to all factories to be used in it are required. This includes the creation of certain factories using factory builders, e.g. a display factory (and associated workspace). The set of factory classes and interface references relevant for a session is referred to as a *world*.

By selecting to *create a world*, a world structure is returned to a client. By passing the world as a parameter in subsequent communication with the configuration service, the client can avoid having to know details of specific factory organisation.

5.6.2 Object Activation

The configuration service allows active objects to be created to process specific data objects. There are three varieties of the activate operation for

- *source*
- *sink*
- *other presentation objects*

Each requires a world reference, a reference to a presentation database object and a callback reference which the presentation object will use for event notification. An interface reference is returned for all successfully activated objects.

In the case of source objects, the configuration service retrieves the type information for the referenced data object and uses the platform database query methods to determine the correct source class. It then contacts the relevant factory and creates the object, passing it the class, data and callback reference. Consider the example from section 5.5.1 of object `Film`. The data originates on the video repository and is of type Pandora so its source object is of class `Source_Video_Stored_Pandora_VideoRepository`. The configuration service uses data from the presentation database and refers to the platform database to work this out and select the correct class of presentation object.

Sink objects are instantiated using the same principles except that a name is passed to enable a data object to be created automatically in the context. This object references the stored output from the sink object. Regarding the class of sink object selected, if the source object accesses data from a storage service, then the sink object will write to this service; if the source object accesses live data then the sink object class will be determined by a policy stored in the platform database.

For general presentation objects the class of the object to be created is passed, e.g. `Video-Display`. The configuration service retrieves the data object type information and checks with the platform database that the class name is valid. If it is valid then the required sub-class to process the data can be determined (using platform database queries) and activated, e.g. `VideoDisplay_JPEG`.

5.6.3 Automated Event Registration

After a presentation object has been activated, the configuration service informs it of events it must monitor for using the object's event registration interface. The configuration service uses platform database information to list all event types which are relevant to the class of active object. Then it determines, for each event type, any relevant events marked on the data object. When registering events, the configuration service passes the callback reference passed by the client.

5.6.4 Type Checking and Automatic Connection

Obviously incompatible objects cannot be linked together. As an example, using the script language it is not legal to specify the following:

```
VideoSourceObject | AudioProcessingObject
```

It is possible to instruct the configuration service to check that the output type of one object is compatible with the input type of another object. The configuration service is passed the classes and interface references of two activated objects. As described in 5.3.1, information is stored in the platform database relating to input and output details of presentation object classes. If the formats of the objects match, then the configuration service uses the class information supplied to determine (through look-ups with the platform database) the methods required to connect the objects. It then automatically connects them.

5.6.5 Automatic Object Insertion

When connecting presentation objects of certain classes the configuration service can activate and insert other objects. In some cases the client is informed that this has been done and is returned class details and interface references for the created objects. An example scenario is inserting a synchronisation object into a continuous media configuration, up-stream of a display object. In other cases the client is not informed, e.g. when converter objects are inserted (see section 5.5.5.2). The platform database stores rules which specify situations for object insertion. It is thus up to an application developer to manager his/her own policies.

5.7 Prototype Implementation

The design of the object support layers has been illustrated. A full implementation of this design was constructed using the Cambridge multimedia environment as a testbed. This section describes what facilities were used to build services and what medium, format, device and presentation object types were supported.

In section 5.7.1, the data characteristics of the environment are described. The decisions as to what support to provide in the prototype platform are outlined in section 5.7.2. In order to build the object support, certain programming and system support facilities are required. These requirements and implementation decisions made are introduced in section 5.7.3. Section 5.7.5 discusses the facilities used to build the databases. Section 5.7.6 describes the arrangement of the prototype active objects.

5.7.1 The Experimental Environment

As has already been described in section 2.2, multimedia environments are characterised by a wide range of heterogeneous services. Some components provide a similar service despite having different control characteristics.

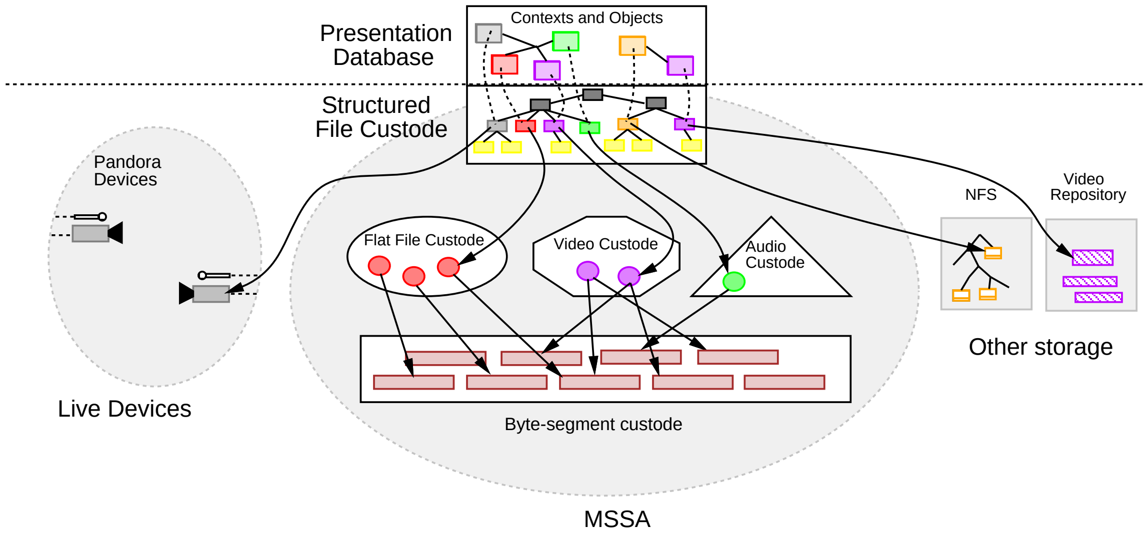


Figure 5.7: Presentation database and experimental environment

The environment in which the prototype platform was constructed is a multimedia environment with a rich variety of devices thus yielding a large degree of heterogeneity. The study shown below has been divided into the major data domains. There are a number of live devices and storage services associated with these. The following main components of the environment are illustrated in figure 5.7:

- *The traditional services* of a distributed environment are available including graphical workstations, connected by a network, and network storage services. Workstation run the UNIX operating system, are connected using an Ethernet and use NFS for filing.
- *The Pandora system* [**Hopper90**] consists of a series of workstations each with camera, microphone, capture and display facilities on top of the usual workstation functionality. A prototype repository has been constructed for the storage and retrieval of Pandora audio and video sequences. Workstations are connected to a high speed ATM network.
- *The Multi-Service Storage Architecture (MSSA)* [**Lo93**] uses byte segment custodes, at the physical layer, to store all data. Built above this, at the logical layer, typed file custodes provide traditional flat files, structured files and continuous media files. Services have been built using a locally developed micro-kernel and ATM networks for data transport.

With reference to the experimental data environment, it must be possible to model objects which originate from live Pandora devices, the video repository, the MSSA or NFS. Device-specific details should be masked. It should be possible to activate and configure presentation objects to manipulate and display information referenced by data objects. If new devices are added to this environment it must be possible to represent them.

5.7.2 Implementation Decisions

Object support within the prototype IMP platform is targeted at the experimental devices outlined in the last section.

5.7.2.1 Data Objects

Support has been provided for data objects of medium types text, still picture, audio, video and audio/video.

Text and pictures are currently supported on device types `NFS` and `MSSA`. Text is supported in `ASCII` format. Supported picture formats are `GIF`, `TIFF`, `XBitmap`, `SunRaster`, `JPEG`, `PM` and `PBM`. Events supported on text are `Click` and `Hyperlink`. Events supported on picture are `Click` and `ClickArea`.

Stored audio and video are supported on device type `VideoRepository` in `Pandora` format. Live video and audio in `Pandora` format are supported from device types `Camera` and `Microphone`. Custodes within the MSSA are under construction and will be supported as they emerge. Event types supported on video are `MarkedFrame`, `Click` and `ClickArea`.

5.7.2.2 Presentation Objects

To support the available facilities, source, display and synchronisation components have been developed. These components can access, synchronise and display all supported medium, format and device data object type combinations. They also have the facility to monitor for all supported event types. The display objects were built to run above the X window system. The organisation of factories to manufacture objects of these classes is described later.

5.7.3 Programming and System Support

In order to implement the object support layers, the following programming and system support facilities are required:

1. *Object programming support*
   A programming language which allows objects to be defined is necessary. The facilities of components within the object support layers are accessed via interfaces. The implementation of the component is encapsulated within the object. Using this method, unnecessary details can be masked from the application developer.

2. *Remote invocation*
   The ability to distribute object services is called for. Facilities for invoking object methods in other address spaces are thus needed.

3. *Persistence*
   In order to store information about the organisation of the environment and references to distributed data items, a means of persistently storing object information is required (as described in section 5.4.4).

4. *Concurrency support*
   Presentation objects are realised as active objects which require facilities to realise their autonomous threads of control.

5. *Stream, QoS and real-time support*
   Abstractions which represent streams are desirable to simplify the programming of continuous media presentation objects. It is useful to be able to specify processing and delivery guarantees for streams.

Requirements 1 and 2 are needed by every component implemented in the platform. The ANSA testbench, which also provides facilities for requirement 4, was used to this end. It is described in section 5.7.4.

The Structured File Custode (SFC) was used to realise requirement 3 in implementing the database services. It is described in section 5.7.5.2.

Research is continuing into appropriate ways of specifying and supporting requirement 5 [**Nicolaou90**, **Coulson93**, **Campbell93**].

5.7.4 The ANSA Testbench

The ANSA testbench[1], in conjunction with the C programming language, was used to build all components documented in this and the following chapter. ANSA provides the following facilities:

- *Abstract data typing*
  ANSA provides two languages. The first, IDL, is an interface definition language defining interface names, types and methods. Methods are defined as procedure handles which include passed and returned parameters. The second language, PREPC, is embedded in C language source code. Implementations of the methods specified in IDL must be provided. Using this mechanism objects can be implemented[2]. Invocation of methods on objects is possible, using PREPC statements, if an interface reference for the object is obtained. Dynamic creation and destruction of objects is possible.

- *Remote Procedure Call*
  ANSA objects can run within different processes and on different machines, since invocations can be realised as Remote Procedure Calls (RPC). Interface references can be exported to a name service known as the *trader*. It allows these interface references to be imported by processes which require them.

- *Tasks and threads*
  ANSA provides component concurrency through threads and tasks. A thread is an independent execution path through a sequence of operations while a task is a virtual processor which provides a thread with the resources it requires

[1] Version 3.1. of the testbench was used [**ANSA90**]

[2] ANSA testbench v3.1 is strictly object based rather than object-oriented although forms of inheritance can be supported using the `IS COMPATIBLE WITH` IDL definition

5.7.5 Building the Databases

This section describes how the prototype databases were implemented.

5.7.5.1 Platform Database

Changes to the platform are likely to occur reasonably infrequently. The prototype platform database allows information to be added statically at session start-up (in a similar way to the X resource database). However, the presentation database requires dynamic, possibly fine-grained, alteration of persistent information. The structured file custode was used to this end.

5.7.5.2 The Structured File Custode

The structured file custode (SFC) [**Thomson90**] is a prototype service within the MSSA (sections 2.2.5,5.7.1). This service allows user-defined storage structures to be created. It provides the following two abstractions for storing data:

- `BYTESEQ` - Byte sequences are abstractions of flat files (as used in traditional file systems such as NFS).
- `SSID` - Storage service identifiers, as described in section 2.2.5, can be used to reference any object within the MSSA. This allows composite multimedia objects to be defined by referencing distributed data items.

These storage types can be combined into structures using the following operators:

- `SEQ` - Sequences can be used to represent a collection of SFC structures of the same type.
- `REC` - A record can be used to represent a collection of SFC structures of different type.
- `UNION` - A union can be used to specify a number of possible structures, one of which can be used at run-time.

A definition language can be used from within C programs to specify structures. An example to specify a sequence of records containing a byte sequence and SSID is

```
SEQ[0,REC[BYTESEQ[0],SSID]]
```

The first parameter of byte sequences and sequences is used to specify the field length. A '`0`', as in the example, specifies a field of variable length.

The SFC interface provides methods to dynamically create, read, write, copy, append, truncate and delete parts of structures. A *path* must be specified in order to access specific fields within structures. For example in the above structure, to access the SSID within the third record in the variable length sequence requires the path `(2,1)`[3].

When a structure of any kind is created, an SSID is returned. This SSID can be included in other structures thus enabling structures to reference one another.

The ability to apply different access rights to different components of structures is under investigation [**Sultana92**]. The SFC is also provided with garbage collection facilities. These find unreferenced data and delete it thus freeing space.

In the next section specific examples related to the management of data objects using the SFC are illustrated.

5.7.5.3 Presentation Database

Persistence is implemented using the SFC and this section partly aims to illustrate the appropriateness of such a service for representing composite multimedia data. Figure 5.7 illustrates the design and principle behind the presentation database, with relation to the experimental environment.

The SFC is used to store the persistent data associated with data objects. An illustration of how the SFC's structuring facilities can be used to support such information is shown below and illustrated in figure 5.8.

```
REC[BYTESEQ[0],
    SEQ[0,BYTESEQ[0]],
    SSID,
    SEQ[0,REC[BYTESEQ[0],
              BYTESEQ[0],
              SEQ[0,BYTESEQ[0]]]],
    SEQ[0,REC[BYTESEQ[0],
              BYTESEQ[0]]]]
```

Methods which create and access the data use the SFC interface and specify paths with which parts of the data can be accessed individually.

When adding objects, a data reference encapsulates an SSID which points to the data. For non-MSSA objects which cannot be represented by an SSID, a variable length sequence of byte sequences (of the form `SEQ[0,BYTESEQ[0]]`) is created. This is used to store the fields of the instantiated data template, e.g. for the video repository these would be a server address and a pathname. The SSID returned when this structure is created is used as the data reference. An `ObjectID` also represents an encapsulation of an SSID.

As illustrated in figure 5.8, attribute information is just stored as a sequence of records containing byte sequences to represent the attribute and its value. Events can be added to data objects as part of a variable length sequence of records containing event information.

Contexts are implemented as a variable length sequence of SSIDs. Each SSID references a

[3] The first branch in the sequence is represented by 0, the second by 1 and so on.

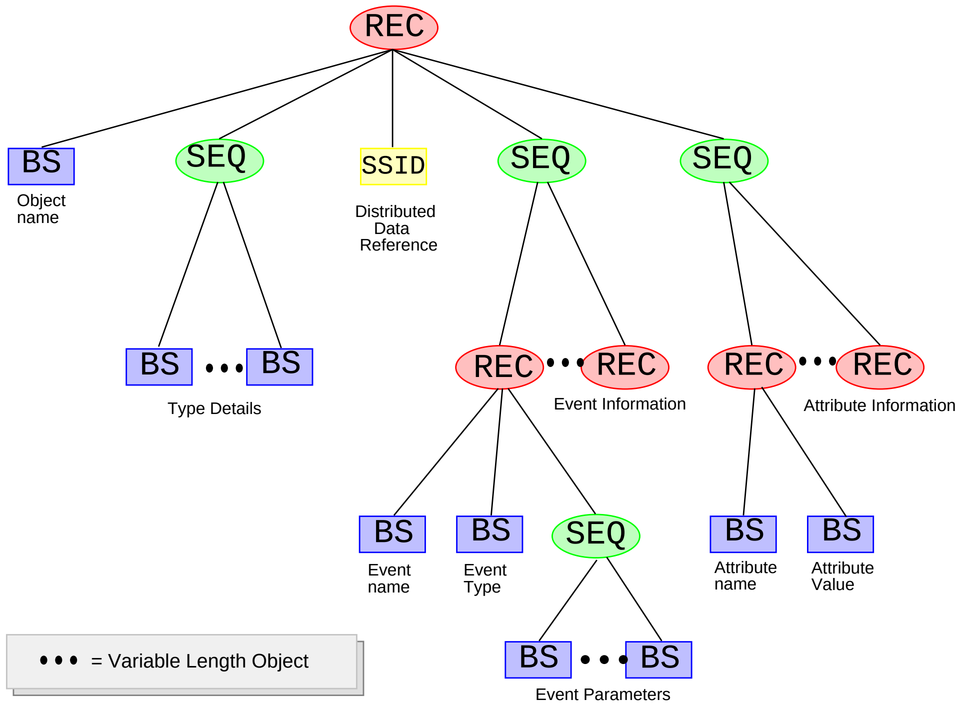


Figure 5.8: SFC representation of a Data Object

data object. Likewise, the top-level database structure is a variable length sequence of SSIDs each of which represents a `ContextID`.

Methods to retrieve information by name require algorithms to search through a structure for the relevant field. Cloning objects involves copying the entire object record structure and returning a new `ObjectID` representing its SSID. Sharing an object involves retrieving the `ObjectID` from one context and storing it in another without deleting it from the first.

5.7.5.4 The Benefits of using the SFC

The SFC enables structured data to be created and accessed at either a course or fine granularity. Fields of variable length allow dynamic alterations in the size of object data. Using SSIDs, these structures can reference one another or reference distributed data objects within the MSSA. Other data references can be synthesised using reference meta-data.

The SFC has other benefits such as its automated garbage collection. Fine-grained access control can be used to support privacy mechanisms for group editing within a context.

5.7.6 The Prototype Active Objects

The prototype active objects were built using ANSA for object and concurrency facilities. They support all the medium, format, device and event combinations described in section 5.7.2.

5.7.6.1 Implementing Presentation Control

Text and picture source objects use NFS and MSSA libraries to retrieve files. Text display objects allow fine-grained manipulation, such as highlighting marked words. Picture display objects interpret individual formats and display them as a bit-mapped picture. These objects have methods to flip, rotate and smooth pictures.

Video and audio source objects use Pandora library functions to control devices such as the repository, cameras and microphones. Stored video and audio source objects take advantage of the repository's abilities for altering the rate of data access (for fast and slow operations), for pausing and resuming the data access and for random access within a video file. Video display objects were constructed using existing Pandora X libraries. The X window system is used to control the geometry and layout of the video display window. The X server has been extended to send geometry and layout control messages to the Pandora box controlling video data display.

Synchronisation objects provide intra-stream synchronisation to remove jitter from audio and video streams. Initially within the IMP prototype, the synchronisation provided by the Pandora system was used. The Pandora box design includes facilities for the buffering of streams of input to remove jitter[4] [**Wray89**]. However, it may be that some continuous media systems do not support synchronisation at the infra-structural level. Software developed as part of a related project [**Sreenan93**] provides a factory for synchronisation objects known as *stream agents*. These perform intra-stream synchronisation for defined formats of continuous media data. Arriving samples are put into an elastic buffer and are sent out at regular intervals based on the characteristics of the format type.

5.7.6.2 Implementing Connection

Text and picture source objects are passed the interface references of objects to communicate with. They retrieve files of data and pass them in tokens to the next object. Video and audio source objects use the Pandora library calls to inform boxes to connect to other boxes. Any presentation objects, such as stream agents, which need to manipulate Pandora data must use a transport level (in this case MSNL [**McAuley89**]) connection to the relevant box.

5.7.6.3 Implementing Quality Control

No QoS negotiation services were available for either operating system or network in the prototype environment although some research has recently been done into these issues within the environment [**Crosby93**, **Hyden94**]. The work illustrated in this chapter takes into account these requirements although the underlying support within the environment is not yet available. The presentation objects developed within the prototype platform use the proprietary facilities available within the environment.

Video source objects were able to use the ability of Pandora devices to adjust the frame rate and image resolution quality in transmission and display. The MSSA provides a ticketing mechanism which guarantees data delivery for a session. A ticket must be negotiated with the byte segment custode. Also, stream agents allow a specification of tolerance to jitter.

[4] Only initially implemented for audio jitter smoothing

5.7.6.4 Implementing Event Functions

All event monitoring functions associated with display components were *referred* to the X window system. This has the ability to call back an application by entering a specified routine if a specific event occurs. Examples of events it can monitor for are windows being clicked on, entered or left. Client data can be attached to specific X callbacks so that the event can be identified when it occurs.

For all display components, events of type `Click` and `ClickArea` are added to event monitoring lists. When the display component `show` method is invoked, they are added as X callbacks. If the handler routine is entered then all `Click` events are notified immediately. For `ClickArea` events, the event monitoring lists must be consulted to see if the coordinates of the click was within the specified area.

Hypertext display components implement the `Hyperlink` event type. Event monitoring tables for these events are kept in word order. When the text is being displayed, if the current word is a registered event then it is highlighted and an X callback is attached. If it is clicked on it notifies the required event.

In the initial prototypes, frame boundary event monitoring was provided as part of video source objects. Marked frames were detected through communication with the source Pandora device. Checking for every marked frame every time a frame is processed is impractical. The event monitoring table is thus put into frame order so that events are checked relative to the frame currently being processed. Stream agents provide frame boundary checking as part of the synchronisation process. Every video frame which is synchronised can also be checked to see if it is a marked frame.

5.7.6.5 Prototype Factory Arrangement

In the prototype implementation (as shown in figure 5.6[5]) factory sites exist as ANSA capsules which are provided with tasks (processing resources) as required. Factories manage the creation of source, synchronisation[6] and display objects. A display factory is required per application as it creates an associated workspace.

[5]The person factory shown is described in section 8.2.2

[6]The synchronisation service (as shown in figure 5.6) can perform inter-object synchronisation and is described in the next chapter

5.8 Conclusions

The object support layers aim to mask out the complexities of a multimedia environment. Information regarding the available facilities is stored in a platform database. Information in this database can be altered, when the functionality of the platform changes, and queried by clients.

The data modelling interface is realised by a presentation database. This provides methods to store, retrieve and manipulate data objects and contexts. Operations must be validated for type-correctness with the platform database.

Presentation objects, which manipulate multimedia data and notify events of interest, are implemented within an active object framework. An active object has data, methods and autonomous threads of control for asynchronous operation. Each object must perform timely execution of its presentation function. It provides methods for adjusting the flow and quality of presentation at run-time. To initialise an object, methods are provided for event registration and interconnection with other objects. Active presentation objects are created by factory objects within sites which can be distributed within an environment.

A configuration service provides facilities to assist a client in the creation and connection of presentation objects without that client having to know their specific implementation details. Source presentation objects, to access the multimedia data referenced by particular data objects, can be instantiated. Objects to perform presentation operations on this data, and sink objects to store the result, can also be created. Configuration of any compatible presentation objects, subject to type-checking, can be performed by this service. In all of these operations a client does not have to know details of factory organisation.

The feasibility of constructing such services is demonstrated by the prototype implementation, constructed within the IMP platform.

Chapter 6

Presentation Services

This chapter describes how *presentation services* use the object support layers to realise authored presentations. A modular design for implementation flexibility is discussed. The implementation approach used in the IMP platform is then described.

6.1 The Role of Presentation Services

Presentation services provide environment-independent composite presentation functions, realised using the object support layers. In implementing authored presentations, the roles of presentation services are

- Setting up configurations of presentation objects
- Monitoring for the firing of event rules
- Event-driven invocation
- Group invocation
- Synchronisation between presentation configurations
- Communication with a controlling application

In order to realise a presentation, the relevant script must first be interpreted. A composite active object, known as an active context is instantiated using script information. This is responsible for managing the above responsibilities. To achieve this it creates and interacts with objects within the object support layers. An active context provides a presentational unit of concurrency. Many such contexts can be active at once under the control of a single application.

6.2 Overview of Presentation Services

Active contexts are divided into a number of modules, as illustrated in figure 6.1.

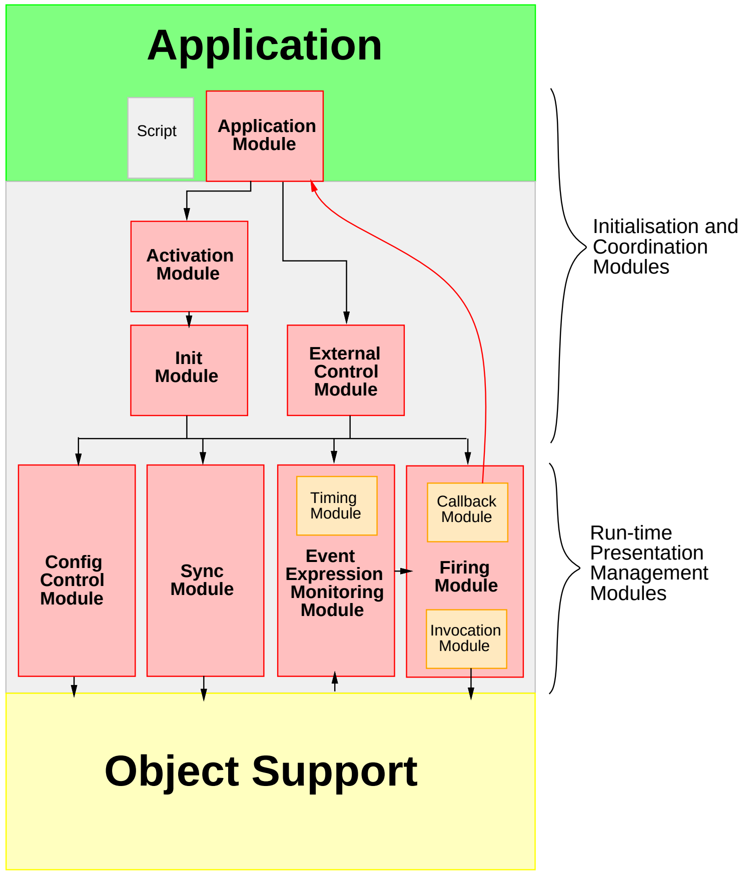


Figure 6.1: The modules which compose active contexts

6.2.1 Presentation Management Modules

The modules concerned with controlling a presentation at run-time are

- *Configuration control module*
  This module activates entire configurations as specified in scripts. Section 6.3 outlines how the configuration control module uses the configuration service to realise the required configurations.

- *Synchronisation module*
  The synchronisation module manages the realisation of run-time synchronisation between data presentation configurations. During presentations, the synchronisation module must ensure that the synchronisation relationships are adhered to. Section 6.4 discusses how the functions of the synchronisation module can be realised as a synchronisation service to provide synchronisation between presentation objects and how such a service can be initialised.

- *Event expression monitoring module*
  This module monitors for the detection of event expressions. Temporal delays in event expressions (see section 4.3.3.7) are realised through the timing module. Section 6.5

illustrates techniques to realise the event monitoring and timing modules and how facilities could be provided for general event monitoring and time services.

- *Firing module*
  This module is notified of the detection of any event expressions. It uses the invocation and callback modules to perform relevant object invocations and application callbacks. Section 6.6 examines how the firing, invocation and callback modules manage the firing of invocations and callbacks associated with rules.

6.2.2 Initialisation and Coordination Modules

The modules concerned with initialisation and external coordination of an active context are

- *Application Module*
  This module implements the context tools interface described in section 4.4. It manages all application level functionality such as ordering the activation of a context and communication with it. It also handles callbacks from an active context by entering routines specified by the user.
- *Activation module*
  This module is concerned with analysing a script and using it to initialise active context state.
- *Initialisation module*
  This module provides an interface for initialisation of the active context state. It performs database lookups, sets up data structures and passes presentation information to the relevant presentation modules.
- *External control module*
  This module manages run-time requests from a controlling application such as external invocations and events.

The activation of a context is started by the application module and is managed by the activation and initialisation modules. This process is described in section 6.7.

Section 6.8 describes the application module facilities for interacting with active contexts at presentation time via the external control module.

6.2.3 Implementation Decisions

The modular approach allows different techniques to be employed in the realisation of these modules. For example they can exist within the address space of the same service, as part of cooperating distributed services or all within the application address space[1].

Implementing these modules as separate cooperating services allows one to capitalise on the potential concurrency in aspects of the presentation. It also allows different functional

[1] This is not to be recommended due to the overheads of some module functions, such as event expression monitoring

approaches to be investigated without affecting the overall architecture. New presentation modules may be required at a future point and can be integrated without difficulty

In section 6.9, the style in which these modules were realised in the prototype platform and tools used to built them are outlined.

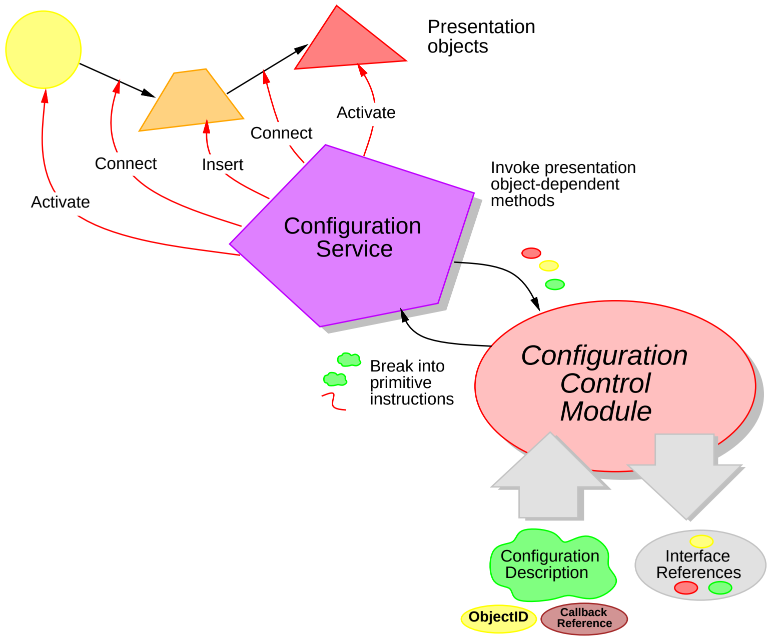


Figure 6.2: Configuration control module

6.3 Configuration Control

The configuration control module is responsible for creating entire configurations (see figure 6.2). The configuration service interface to the object support levels (section 5.6) is used to create the necessary presentation objects. A *world* is first created for use by the presentation. This is passed as a parameter in subsequent operations, to allow the configuration service to identify the factories associated with this presentation.

One issue which should be explained is the separation of the configuration service from the configuration control module. The configuration service is located within the object support layers and can activate single objects and connect pairs of objects. It accesses platform policies stored in the platform database which specify such situations as when to insert presentation objects into configurations. It thus provides platform-dependent functionality and type-checking of a fine granularity. The configuration control module is an environment independent service which can be passed entire configurations, in some format, which it will then activate. It does not need to know system-dependent details of how to connect specific presentation object classes together. Configurations need not be sequential, as provided for in the authoring model. They can involve additional complexity, such as multiple branches from one component.

Each configuration is passed as a collection of presentation object classes and an `ObjectID`. In the case of the authoring model presented in chapter 4, configurations will be passed as lists. The configuration control module goes through them in order, one element at a time. For each item, it uses the configuration service to try to activate a relevant object. It passes the reference to the world, the presentation object class and the `ObjectID` for the data object. The client of this module must provide a callback reference which is also passed (see section 6.5.3), with which the activated object will notify event occurrences. Each activation operation is successful if a presentation object class is valid. The configuration service will automatically register any relevant events with the active object. It returns an interface reference for each activated object.

In order to connect presentation objects together, the configuration control module passes object details (classes and references), two at a time, to the configuration service in an attempt to connect them. The service checks that the connection is valid and, if so, connects them. If, during connection, any additional objects are inserted (see section 5.6.5) then the classes and interface references are returned by the configuration service.

The configuration control module returns its client an ordered collection of presentation classes and interface references, representing the activated configuration.

6.4 Synchronisation

The duty of a synchronisation module is to manage inter-configuration synchronisation in the presentation of continuous media.

6.4.1 The Problem of Synchronisation

Explicit synchronisation between the presentation of media objects can be required in situations such as a soundtrack synchronised to a video sequence (lip-sync'). Despite the streams of information controlled by presentation objects being started at the same time, the data is not necessarily synchronised. The nature of packet networks means that different delays can be introduced between the transmission of individual continuous media segments. This problem is known as jitter and can result in frames arriving late and thus being presented out of synchronisation. If streams need to be synchronised together then differences caused by jitter can create skew, a temporal mismatch in the presentation of media.

6.4.2 A Synchronisation Service

Several solutions have been suggested to solve this problem [**Anderson91**, **Blakowski91**, **Coulson91**]. The approach subsumed by this design is proposed in [**Sreenan93**]. This technique puts forward a general purpose synchronisation service, which both integrates easily with the object support layers and is in-keeping with the modular design principles of presentation services. It achieves synchronisation through run-time interaction with synchronisation presentation objects in the relevant configurations. Despite the obvious concern of jitter removal, this is a reason why such synchronisation objects could be inserted automat-

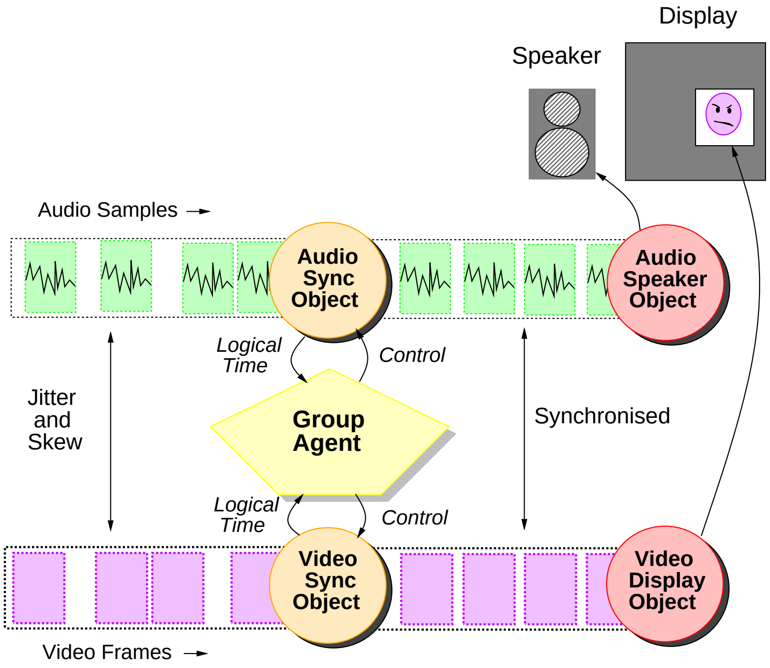


Figure 6.3: Synchronisation using group agents

ically into a configuration (see section 4.3.2.2). This scheme of inter-object synchronisation has some influences on the implementation of synchronisation objects.

Synchronisation objects (such as stream agents, already described in section 5.7.6.1) manage jitter removal within single streams. Each synchronisation object must maintain logical stream time information indicating the stream's status in presentation. To implement inter-stream synchronisation *group agents* are created. At intervals, related synchronisation objects report their logical times to a group agent. By controlling the synchronisation objects, the group agent ensures the skew between stream times does not exceed specified bounds. For example, a group agent could be created to synchronise Pandora audio segments to frames of JPEG video. The illustration of this concept is shown in figure 6.3.

A synchronisation service manages group agent creation. A service instance usually runs in the same factory site as synchronisation objects are manufactured. This makes communication between group agents and synchronisation objects more efficient. When group agents are created, they are passed the interface references of relevant synchronisation objects. Using these, group agents can initialise the synchronisation objects so that they report to them at presentation time.

6.5 Event Expression Monitoring

The event expression monitoring module is responsible for checking event occurrences from a presentation world against expressions specified by the user.

6.5.1 Initialisation

In order to initialise the state of the event monitoring module it must be passed details of

- *The event domain* - which it maintains in an event domain table
- *Event expressions within that domain*

6.5.2 Finite State Automata

The event expressions are stored internally as finite state automata (or finite state machines) [**Manna74**, **Gehani92**]. These are composed of nodes, representing states, and arcs, representing transitions between these states. In this model, the arcs are labelled with events. At run-time, the occurrence of any event in any state has an effect on an automaton- either causing it to enter a new state or loop in its current state. If an automaton enters an accepting state then the event expression is detected. The operators illustrated below control the construction of finite state automata. They return the caller a handle to the machine, which can be referenced in subsequent calls. The operators construct automata in the following ways:

- *Primitive events*
  Primitive events, such as marked object events, system events and external events, create a finite state automaton with a non-accepting start state. If the required event occurs, it enters an accepting state. On the other hand if an event occurs which is not the required one, it enters a non-accepting state. From there on, the automaton enters either the accepting or non-accepting state. The `any` expression has the effect of creating an automaton that always enters the accepting state.
- `not`
  This operator changes a machine so that the accepting states become non-accepting and vice-versa. The start state remains non-accepting.
- `every`
  This operator replaces the arcs of any accepting states with those from the start state. Thus, if the composite event is detected, subsequent state transitions are as if the machine is uninitialised.
- `after`
  This operator joins two machines (A and B) together in sequence. The arcs from the start state of machine B replace those from all the accepting states of machine A. The new machine has a single start state- that of machine A.
- `and`
  This involves creating an automaton whose states consist of all combinations of the states from the machines of the corresponding sub-expressions. Assume the two automata for the sub-expressions are called A and B, with corresponding sets of states $\mathcal{A}$ and $\mathcal{B}$. The new machine AB has a set of states $\mathcal{AB} = \mathcal{A} \times \mathcal{B} (= \{(a,b) : a \in \mathcal{A} \text{ and } b \in \mathcal{B}\})$. For every arc in A ($a \to a'$), and every arc in B ($b \to b'$) there is an arc in AB ($(a,b) \to (a',b')$).

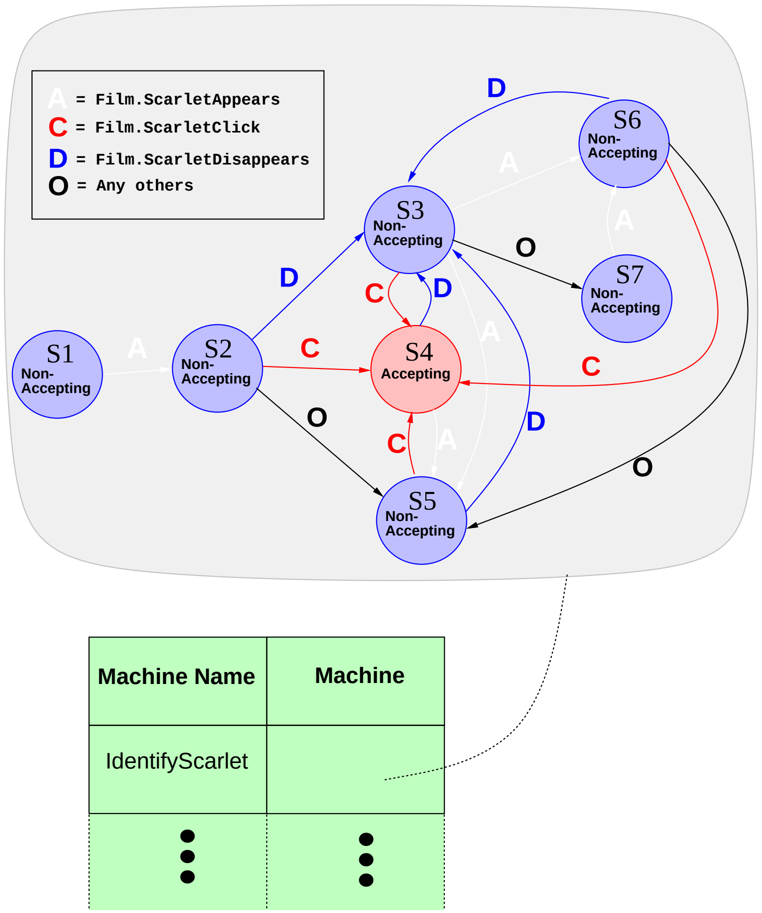


Figure 6.4: Example finite state automaton

- `delay`
  The delay operator sets up a new rule, using the finite state machine it is passed (machine A). Instead of recording details of a client to notify, it records details of a *system invocation* to set a timer. Such invocations provide a way for the system to use event-driven features. This type of system invocation uses the timing module to set a named timer system event (described later in section 6.5.5). The `delay` operator returns a reference to this system event as the sub-expression handle, which can be incorporated into another automaton (machine B). The run-time scenario is as follows: if the expression (machine A) in the newly created rule is detected, the timer is set and, when the delay has elapsed, the system event, incorporated as part of another automaton (machine B), is notified. This process is entirely transparent to the application user.

- *Event rule names*
  This has the effect of creating a copy of the automaton from the referenced rule and returning the client a handle to it. Thus rather than adding the names of event rules to the event domain, the actual automaton for the entire expression is inserted into the new expression.

Figure 6.4 shows an example finite state automaton representing the `IdentifyScarlet` event expression (see section 4.3.3.4).

6.5.3 Handling Presentation Events

The event expression monitoring module provides an interface through which it can be informed of the occurrence of presentation events. This interface must be passed as a callback reference to any objects involved in the presentation which can notify events.

When an event is notified, the parameters must include the object and event instances so that the event can be identified and relevant actions taken. In an extended model, the module may also be able to perform queries on the parameters of occurring events as part of more complex detection mechanisms. Within the local area it is reasonable to assume that the time at which events arrive corresponds to the order in which they occur. In the wider area, timestamps should be associated with events, as the order in which they arrive from different sites may not correspond with the order in which they occur.

6.5.4 Updating Automata

Any event occurrence causes all finite state automata to be updated. For each machine a pointer to the current state is recorded. In every state, a list of possible arcs to traverse on the occurrence of events, must also be stored. If an expression is detected, the event monitoring service informs a specified client.

6.5.5 The Timing Module

To set a time event, the name of the event which is to be notified is given as a parameter as well as the delay and an interface reference on which the module will call back.

To implement temporal events, the processor's real-time clock can be used, to time specific delays. The timing module must keep a list of delays, to notify in order of temporal occurrence. When a certain delay period has elapsed, the time service notifies the caller and waits for the next delay.

The timing module can be implemented as a separate *time service*. Despite the communication delay in timer setting and notification, this is recommended to remove processing delay from the event monitoring module.

6.5.6 A Shared Event Expression Monitoring Service

An event expression monitoring service shared between different applications can be envisaged. Different types of client could register expressions with this service. It could call them back if an expression was detected. The benefit of such a shared service is that it would be a useful tool for building collaborative applications. Events from different contexts and users could be combined within expressions. Also, the service could inform multiple interested parties of event expression detection, at the same time.

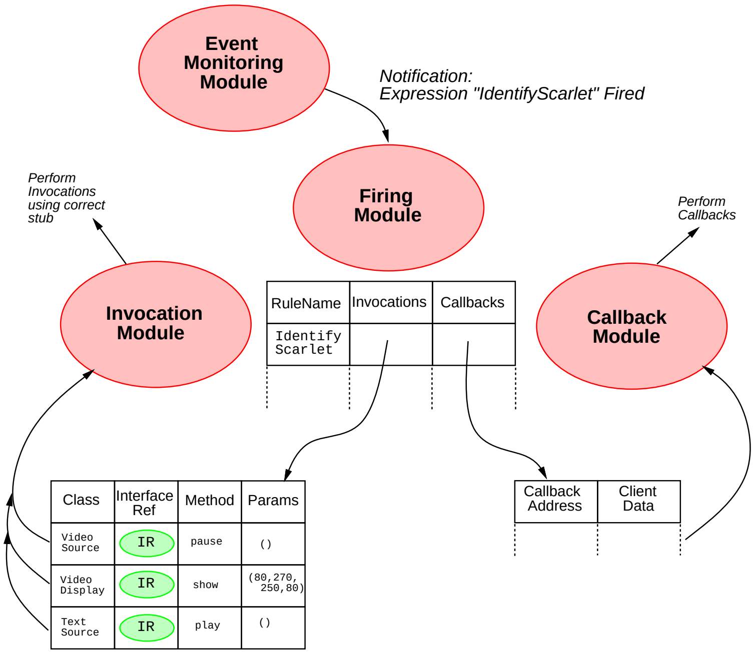


Figure 6.5: An event-driven rule firing

6.6 Firing Invocations and Callbacks

The firing module is responsible for firing event rules. It can invoke a series of operations on presentation objects involved in a presentation and can perform a series of callback operations to the application.

Invocations are stored in *invocation list* structures. These are indexed by rule name and contain fields referencing the class, interface reference, method and parameters for each invocation. Callbacks are stored in *callback list* structures indexed by rule name. These contain fields referencing the callback reference and the client-specified data to pass back. In the event of a notification that a named event expression has been detected, the name is used to look up associated invocation and callback lists. Invocations are passed one by one to the invocation module and callbacks to the callback module. Figure 6.5 illustrates the firing of the invocations associated with the rule `IdentifyScarlet` from section 4.3.3.4.

The invocation module is passed an element from a invocation list structure and is responsible for performing the invocation in a type-correct manner. The invocation module has access to stub routines for all classes of presentation object. Using the appropriate routine for the class and method, supplied with the interface reference and parameter information, it performs the invocation.

The callback module performs an invocation on a callback interface. The client will have created a relevant object to handle the callback (see section 6.8.2) and passed its interface reference as a callback reference.

6.7 Initialising Active Contexts

This section describes how the modules of an active context, which have been described, can be initialised with script information.

6.7.1 Activation

Active contexts are created using the application module. An instance of the application module must thus be active when the controlling application starts up since it provides facilities for starting all other presentation modules. For this reason, the application module is usually realised as a library within the application program. As illustrated later, this also allows it to manage the creation of local objects and to execute user-specified callback routines.

When contexts are activated, an instance of an activation module is firstly created. This is initially concerned with creating the other relevant modules. It can have to call many services to do this, and is returned a series of interface references. The initialisation module requires references to all run-time presentation services so it can contact them and pass them references to any other modules they must communicate with. The activation module also returns an interface reference to the application module. This is used for external client control during presentations (see section 6.8).

The activation module is also concerned with pre-processing and parsing a script and passing the information on to initialise the active context state.

In this model, scripts are crafted as part of applications but are stored as separate named files. An approach was considered in which the parsing process could generate code to be integrated into an application. The script language could be provided as an embedded language, like ANSA PREPC (see section 5.7.4). This would involve providing a language pre-processor, used before program compilation. The activation module would not be required as generated code would be designed to interact directly with the initialisation module at run-time. However, the preferred approach is to keep scripts separate. It allows any named script to be activated on-the-fly, and not just ones for which initialisation code has been generated. In any approach, no data structures can be created by the initialisation module until run-time, and the activation module adds little overhead to this process. An interactive presentation can be prepared for instant display as an application can activate it (section 4.4.1) separately from starting it up (section 4.4.2). Keeping the script in a form which can be interpreted on-the-fly also means that it is possible to send script statements continuously throughout a presentation if this is required.

The activation module is passed the name and location details of the relevant script. This module can be located either with the initialisation module, or as part of the context tools library with the application module.

6.7.2 Pre-processing

A first stage of parsing script statements must convert any user-defined functions into basic script structures. Example uses of this are

- *To instantiate script templates*
  For scripts which specify object parameters (see section 4.3.1), the `<Instantiation Parameters>` must be specified to instantiate them with real object values from the context.

- *To decode user-defined event expression operators*
  A composite event expression specifies an ordering in the occurrence of events which, if it is detected, fires a rule. This ordering must be specified with the minimal operator set of {`and`, `not`, `after`, `every`}. Other operators can be specified but they must be defined in terms of operators of the minimal set. Examples of such operators are given in section 4.3.3.6. During pre-processing these user-defined operators are converted into their constituents, so that they can be used to build event monitoring structures.

6.7.3 Parsing and Initialisation

The parser syntactically analyses the script and reports any errors which occur. It passes the initialisation module details of the parsed grammar rules (see figure 6.6). The initialisation module is concerned with using this information to set up data structures and passing relevant information on to other modules. Semantic checking can be performed at this level but is not necessary as relevant modules can signal errors in referenced object and event details.

Parsing and initialisation can be divided into the processing of the following script language[2] constructs:

- *Context name*
  Details of the context event domain, containing all marked, system and default events are retrieved by the initialisation module and passed to the event expression monitoring module.

- *External events*
  Details of external events are also passed on to the event expression monitoring module.

- *Configurations*
  For each configuration, the initialisation module retrieves the `ObjectID` of the data object and passes it, with configuration details, to the configuration control module. A callback reference, obtained from the event expression monitoring module, is passed for object event notifications. The returned interface references and relevant classes are stored in *configuration lists* (see figure 6.6).

[2] For syntax details see chapter 4

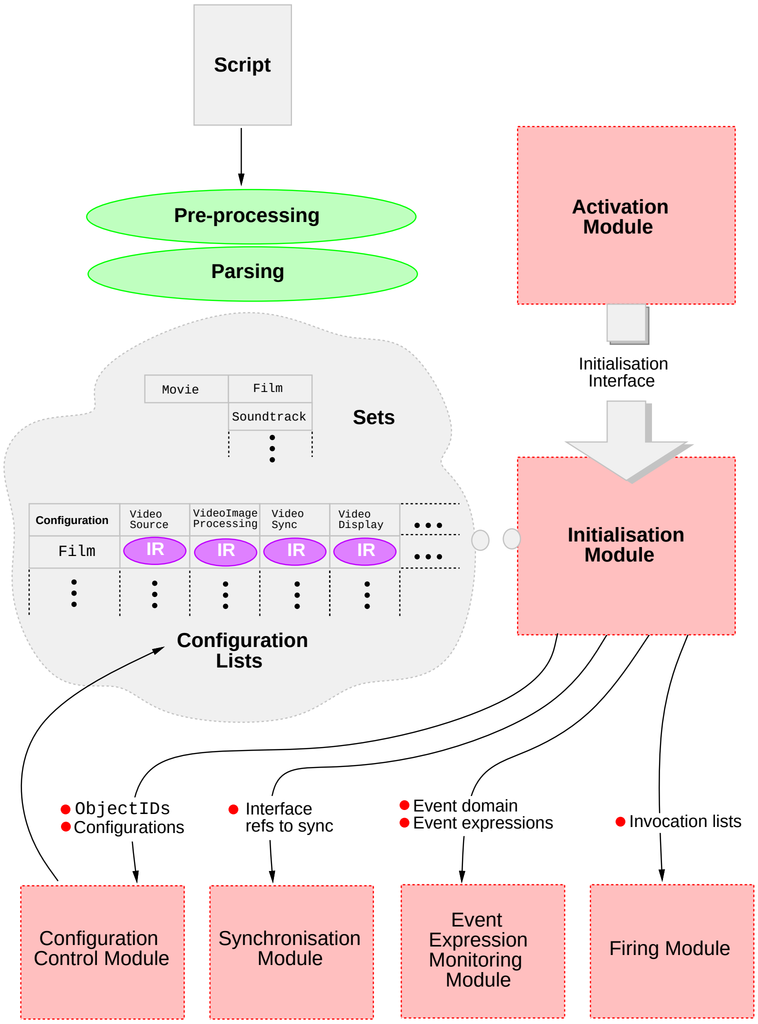


Figure 6.6: Active context initialisation

- *Sets*
  The sub-expressions, which compose sets, are passed to the initialisation module in a post-order manner. The validity of objects referenced is checked and any errors reported. The initialisation module has methods to create subsets representing the result of `union`, `intersection`, `not` and `select` operations. For `select` operations, the presentation database must be queried to find which objects meet the search parameters. For each created subset, a handle is returned to the activation module. This can be used subsequently to define a set or to include the subset as part of a set. When sets are defined the objects which compose them are stored in *set tables*.

- *Synchronised objects*
  The interface references of synchronisation presentation objects from configurations to be synchronised are passed to the synchronisation module.

- *Event expressions*
  The sub-expressions composing an event expression are passed to the event expression monitoring module in a post-order manner. The validity of events referenced is checked

against the event domain table, and any errors are reported. Event operators `not`, `and`, `after`, `every` and `delay` take as parameters the sub-expressions which they operate on and return a sub-expression handle. Sub-expressions can be used to define entire event expressions or can be passed to other operators. The structures built to monitor for event expressions were described in section 6.5.2. Completed expressions are indexed by their rule name.

- *Invocations*
  The interface references and invocation details are passed on to the firing module. If an invocation is to be performed on a set of objects then an invocation is passed for every object in the set. If multicast facilities are available for object invocations, then details should be passed indicating this facility is required and specifying the group invocation to be performed.

6.8 Application Control of Active Contexts

The application module provides the interface to the context tools (discussed in section 4.4). It has facilities for applications to control active contexts at run-time. This module is usually implemented as a library which is included by the application. It sets up any necessary data structures within an application and handles any run-time callbacks (from the callback module). It sends external control requests to the external control module.

6.8.1 External Events and Invocations

On receipt of an external event invocation, the external control module notifies the event monitoring service as if an event had been received from the presentation world.

On receipt of an external invocation, details are passed to the firing module via the initialisation module. This caters for the possibility that the invocation is to be performed on a set of objects.

6.8.2 Application Callbacks

To register a callback requires the name of the rule to attach the callback to, a routine which will be called within the application if the rule fires and a pointer to client defined data to be passed to this routine. The application module creates a structure containing the callback routine and client data information. It passes a reference to this structure as client data when adding the callback.

If a rule fires during a presentation, then the associated callbacks to the application are performed. The application module has an interface through which these are notified. Using the data which is returned, it retrieves the client's callback routine and calls it with the relevant parameters- the active context interface reference, the name of the rule that fired and the client-defined data.

6.8.3 Context Groups

A *context group list* is used within the application to store the interface references of every active context in a group. This can be initialised and added to by the operations `AllocGroup` and `AddToGroup` (see section 4.4.6). When a group operation is used, the single active context operation is called for every member of the group using the parameters provided for the group operation. If an appropriate multicast mechanism is available, then this should be used.

6.9 Prototype Implementation

A prototype implementation of presentation service modules has been constructed as part of the IMP platform. All modules were built using ANSA objects and concurrency facilities. Modules which were required to communicate with other modules outside their capsule were provided with RPC interfaces.

The application module and activation module are both implemented as a context tool library. The C interface to this has been documented in section 4.4. To implement both the pre-processor and the parser, the utilities LEX [**Lesk75**] and YACC [**Johnson75**] were used. LEX provides automatic lexical analyser generation based on regular expression specification. YACC provides automatic LALR1 parser generation based on the specification of grammar rules. The prototype program development environment was UNIX and thus scripts are stored as NFS files. Activated contexts are provided with an RPC interface for initialisation. At run-time, the context tool library communicates with active contexts through the external invocation RPC interface. RPCs are also used to perform callbacks to the application.

The other modules are implemented as separate services. Two possible approaches were available to meet the requirements of the synchronisation module in the prototype environment. Firstly, the design of the Pandora system subsumes inter-stream synchronisation [**Wray89**]. The Pandora format includes time stamps, which can be used to synchronise the presentation of related streams. This is adequate if the streams come from the same source, such as video and audio captured from cameras and microphones attached to the same Pandora box. Time stamps can be added at source and used to synchronise at the sink. However, the IMP system is being extended to support audio and video objects which may be stored on different storage services or originate from unrelated cameras and microphones and may be of different formats. This makes synchronisation more difficult. The second approach available is the prototype synchronisation service (illustrated in 6.4.2) which was implemented as part of the work documented in [**Sreenan93**]. This initially provided group agents to synchronise the Pandora format video and audio stream agents (section 5.7.6.1). As stated, the design of this service is subsumed within the IMP architecture. Despite the addition of support for new continuous media formats and devices, facilities for synchronisation can still be supported using this approach.

6.10 Conclusions

It has been illustrated how presentation services can realise interactive presentations expressed using the authoring model. An active context controls a presentation at run-time. It is composed of modules, each of which fulfils a different presentation function. The design is flexible in that modules can be implemented within the same service or as part of cooperating distributed services.

Modules which fulfil run-time presentation functions have been discussed. Configurations of presentation objects are created by the configuration control module. An approach to synchronisation using group agents to control synchronisation objects has been discussed. Presentation objects are configured to notify events to an event expression monitoring module. Event expression monitoring is implemented using finite state automata for efficient detection. A firing module performs multiple invocations if a rule fires.

Modules for initialisation and coordination of a presentation have also been described.

The feasibility of providing an implementation of this design has been illustrated. A prototype implementation of all active context modules has been built within the IMP platform.

Chapter 7

Experimental Analysis

The purpose of experimental analysis is to ensure that a developed system fulfils its claims and to discover the limit of its abilities. The experimental analysis of the work presented in this dissertation tests its two main aspects- authoring and presentation.

Both authoring and presentation have specific clients. The client of authoring is the application developer. The authoring approach makes it simpler to develop multimedia applications and increases their portability. The client of presentation is the application end-user. An authored application is designed to be presented to and interact with such a user. This experimental analysis thus aims to

- Illustrate that interactive presentations can be built easily and are portable
- Illustrate that a wide range of applications can be supported with the developed facilities
- Demonstrate that the prototype run-time components function correctly
- Allow end users to interact with sample applications

This chapter discusses a suite of experiments which fulfil these aims. Section 7.1 outlines the hardware and software facilities used in these experiments and reviews the main platform services. Section 7.2 describes the experiments themselves and which platform features they demonstrate. Finally, section 7.3 discusses lessons learnt in developing the applications.

Technical details relating to the applications illustrated in this chapter are shown in greater detail in appendices A and B.

7.1 Experimental Apparatus

Figure 7.1 illustrates the experimental apparatus used for the suite of experiments. The host workstation used was an Acorn R140 running the UNIX operating system and using the X window system for window management. It was equipped with a Pandora box with camera, microphone and speaker to allow continuous media capture and display. It had access to Ethernet and CFR networks to connect to storage services, such as NFS, the video repository and the MSSA, or live devices. The object support layers (as described in section 5) manage this infra-structure and the presentation services (as described in section 6) control the flow

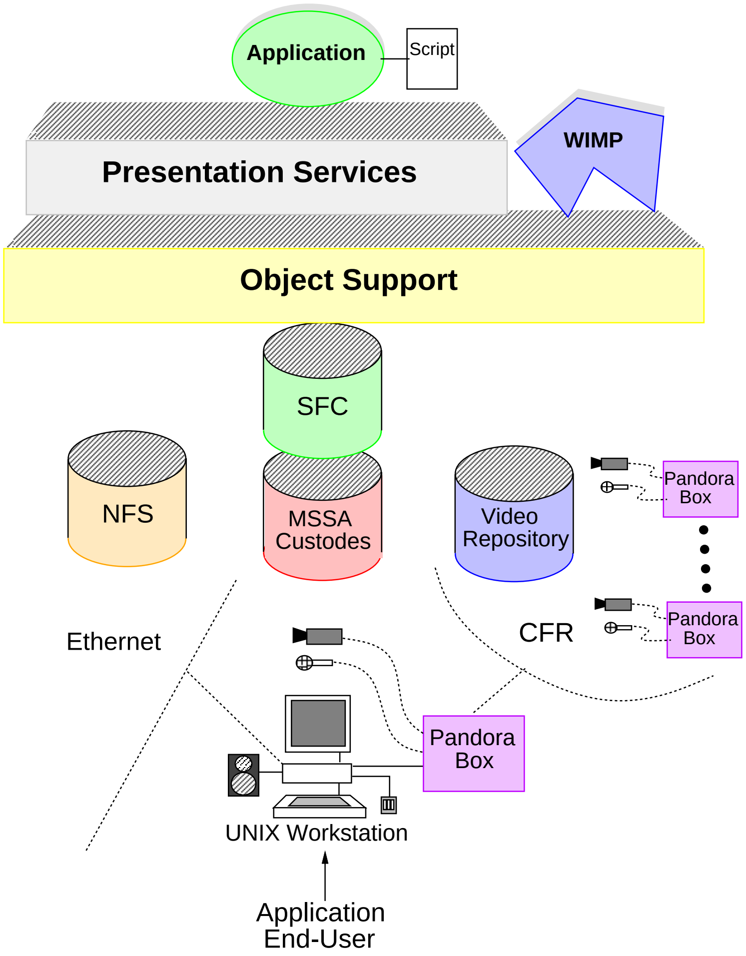


Figure 7.1: Experimental apparatus

of a presentation. They were run, as a series of distributed factory sites and presentation services, on DEC 5133 machines.

A prototype data modelling tool *WIMP* (as described in section 4.2.8) allows direct manipulation of the presentation database. This was used to add objects (see figure 7.2) and to browse the database context (see figure 7.3) in the building of experiments.

7.2 The Experiments

With reference to section 2.1, the three main types of multimedia application are presentational, compositional and collaborative. This section is divided into illustrations of compositional and collaborative application types. The ability to build presentational applications is subsumed by these two types.

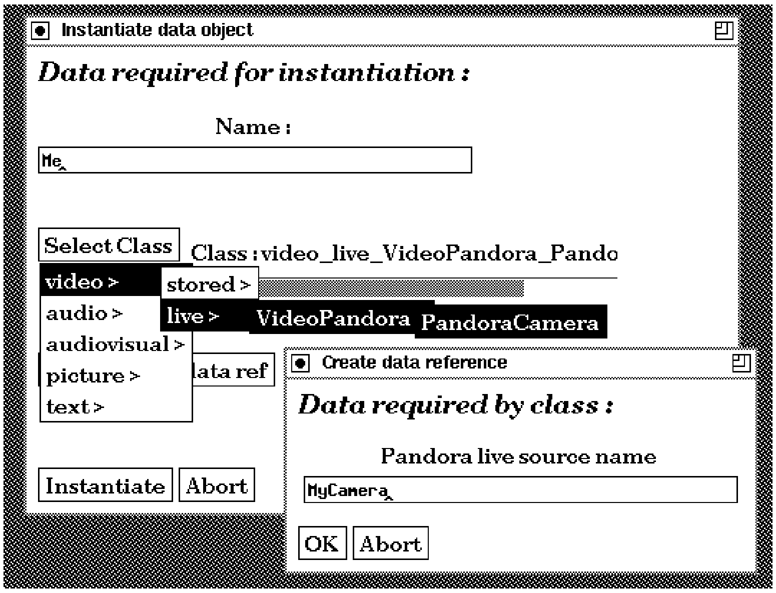


Figure 7.2: Adding a data object

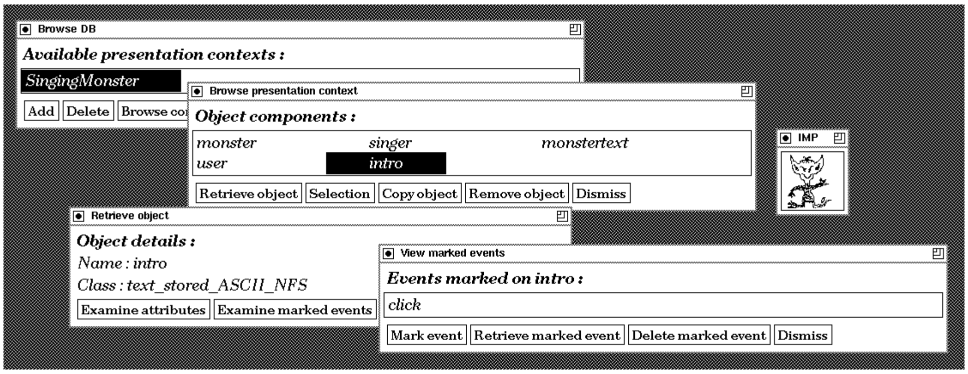


Figure 7.3: Browsing database objects

7.2.1 Compositional Applications

Sample compositional applications developed illustrate

- How media can be mixed together and synchronised in the same presentation
- How cross-referencing mechanisms such as hypermedia links can be supported using event handling
- How interaction and temporal occurrences can be used to control a presentation dynamically

7.2.1.1 Interactive Singing Monster

This application illustrates how text and synchronised video and audio can be used interchangeably and displayed on demand through user interaction. It uses the following object

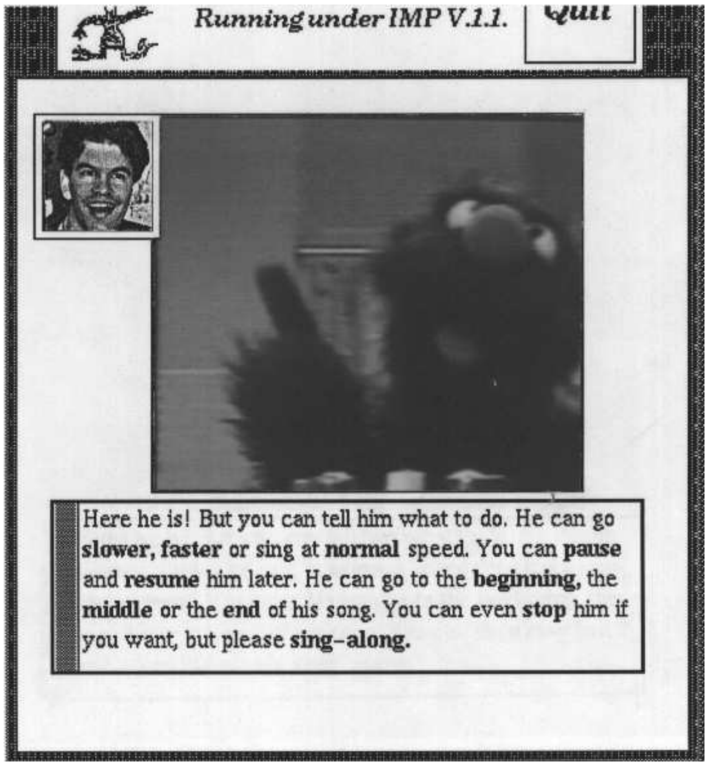


Figure 7.4: Interactive Singing Monster

configurations:

```
monster | AvDisplay ;
intro | TextDisplay_Hyper ;
monstertext | TextDisplay_Hyper ;
user | VideoDisplay ;
singer | Speaker ;
```

It starts up showing text introducing the Monster, a character from Sesame Street, using the object `intro`. The concept of event handling is illustrated as interaction with objects causes methods to be invoked on other objects. When the Monster's highlighted name is clicked on, video and audio start to play, and a different text object, `monstertext`, replaces the previous one. By clicking on highlighted words in the text it is possible to control the singing monster. These words have been marked in the database as hyperlinks. Control is achieved using event-driven rules to invoke methods on the source audio/video object. Options given in the text are sending the Monster's song to the beginning, the middle or end, playing slower or faster, pausing and resuming or stopping. This kind of facility can be used to built hypermedia applications.

The ability to display multiple media in the same workspace is also illustrated. The user can select a sing-along option which displays locally captured video and audio mixed with the monster singing. The following example illustrates how hypertext event `sing` is used to start up `backingsinger` (synchronised set of `user` and `singer`):

```
rule sing_along = monstertext.sing ->
                {
                  user$Display.show(10,20,60,60);
                  backingsinger$Source.play();
                }
```

7.2.1.2 Laboratory Presentation

This application illustrates how an interactive scenario-based information system can be built up. It shows views around the University of Cambridge Computer Laboratory and information can be retrieved by interacting with the series of scenes. The previous application illustrated how simple event-handling can be used for user interaction. This application uses more complex interaction via composite event expressions. A scenario developed in this application, involving the rule `IdentifyScarlet`, has been used as an example throughout chapters 4, 5 and 6.

7.2.2 Collaborative Applications

These applications illustrate

- Creation and control of active contexts to manage the presentation of each user
- Using callbacks from these contexts to affect other active contexts thus allowing interaction between several participating users
- Policies for controlling Quality of Service using sets
- Simple floor control mechanisms

7.2.2.1 IMP Phone

This application provides a user to user audio/video conference facility. Two active contexts are created to manage each of the users' presentation requirements. These communicate with the same controlling application (see figure 7.5). Two live video and two live audio objects are used. The script template used to define those involved in the conference is illustrated in section 4.3.1. The call recipient is asked if he/she wishes to participate in the call. If he/she does then the caller is displayed on his/her workstation, and a callback is sent to the controlling application which informs the caller's active context to display the other party.

7.2.2.2 Opera Meeting

This application replaces the weekly meeting of Cambridge University Computer Laboratory's "Opera" multimedia research group. The Chairman starts up the application, which will then create an active context for every participant. Each group member is initially asked if they

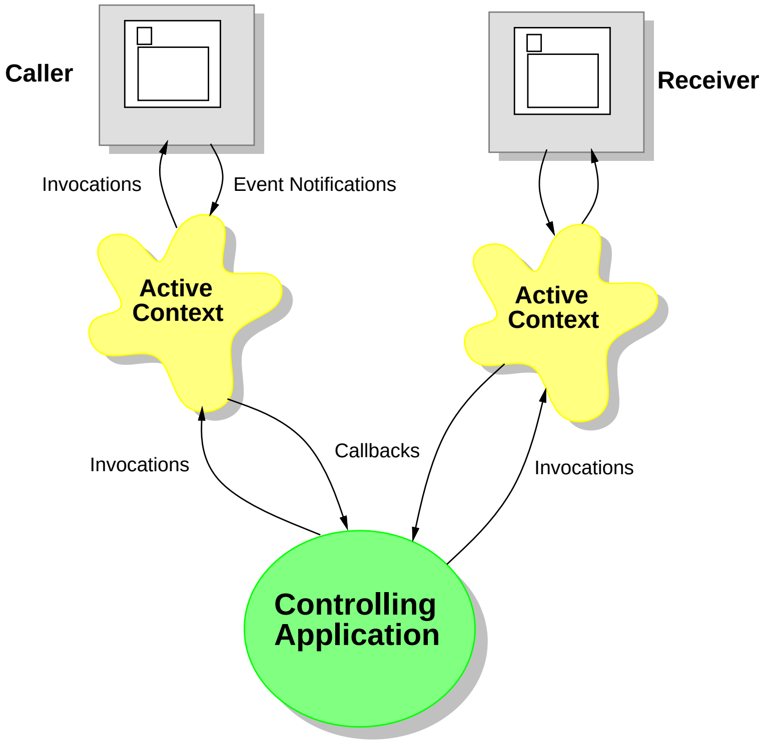


Figure 7.5: Structure of IMP phone

wish to participate in the meeting. They see video of the Chairman who is already 'at the meeting'. If they select the option to join, then they too are added to the workspace of every user. This is achieved by calling back the controlling application which determines which user to add and performs a group update to the active contexts of all conferencees. The controlling application can also dynamically calculate the layout of windows, which is not possible in the script language.

The Chairman's context is different from the others. He manages a simple floor control protocol which uses an icon to indicate the current speaker. The Chairman selects the next speaker by clicking on the relevant participant's window. This issues a callback to the controlling application which performs group updates to every active context to move the icon and turn on the audio from the selected conferencee. It is possible to develop more complicated floor control protocols if required.

Within the presentation database, all objects are marked with an attribute "medium". In the script the following set is defined to contain all video objects:

```
set vid = select(medium=video);
```

If the application receives an external QoS event, a group invocation is performed on this set

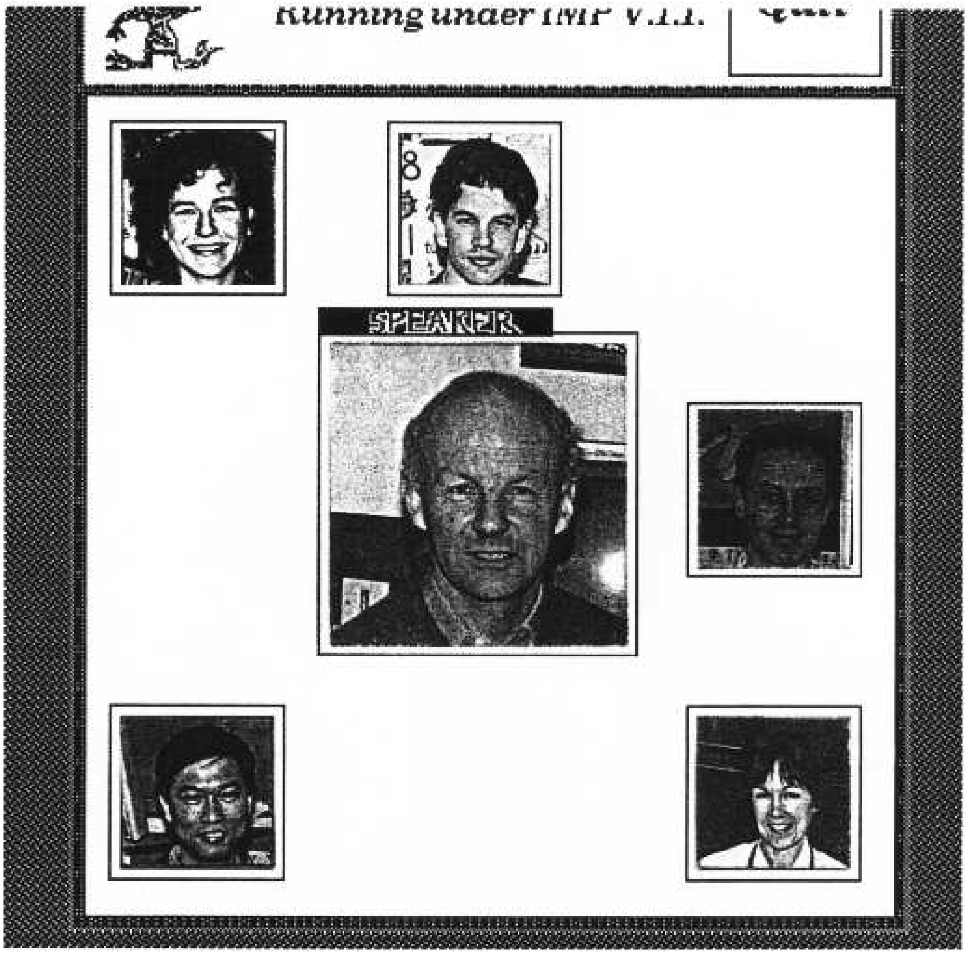


Figure 7.6: Opera group meeting

to calibrate frame rate at 10% of full rate and degrade picture quality to the lowest setting.

```
rule degrade = QoSEvent ->
               {
                 vid$Source.set_rate(10);
                 vid$Source.set_resolution(0);
               }
```

Using techniques developed in the compositional and collaborative applications it is apparent that facilities such as showing an interactive presentation to many users at once are possible. The interaction of a user within an active context can be used to affect presentation in other contexts. It would also be possible to add other collaboration aids such as a shared drawing facility.

7.3 Analysis and Conclusions

The experimental analysis illustrated the following benefits of the IMP platform approach:

- *Ease and speed of development*
  A short script can express a complex interactive presentation. As a comparison, the main file of the Pandora program *XVphone* (see section 2.1.4) has over 3000 lines of C language code. The IMP scripts for IMP phone and Opera meeting were 67 and 40 lines respectively, and the controlling applications were 90 and 350 lines of C code. Similarly, compilation was very much quicker with IMP applications as only the ANSA testbench libraries needed to be included instead of control libraries for devices or services, such as X windows or Pandora. The average executable size with IMP applications is 500 kilobytes as opposed to an average Pandora size of over 1 megabyte. These benefits are possible because all the complexity associated with writing and compiling programs is managed by the platform.

- *Portability*
  The platform separates details of object implementation from the expression of interactive presentation. This enables an application to be ported between platforms. Experiments showed that data objects can be moved between services, such as a text object being kept on the flat file custode rather than NFS, without affecting an application. Thus if the environment alters or the application is moved to another environment, the data items involved can be re-referenced on different services.

- *Support for a wide range of applications*
  It has been illustrated that a wide range of applications can be constructed including desirable features from presentational, compositional and collaborative applications. The ability to extend the platform means that new support facilities can be added if required.

- *Service efficiency*
  The separation of presentation functions between distributed services means that no particular service is a bottleneck to real-time presentation. The major bottleneck in experiments was X window updates on the local workstation, but only due to the comparatively low processor speed of this machine. However, it became apparent that a large amount of finite state machine updates as part of event monitoring could be a time-consuming exercise in highly interactive applications. For use on a large scale, finite state machine optimisations must be investigated.

- *Application usability*
  The experimental applications have been viewed, tested and enjoyed by both members of the Laboratory and visiting academics.

Chapter 8

Conclusions

8.1 Dissertation Review

This dissertation has described investigation into and conclusions reached regarding presentation support for interactive distributed multimedia applications. Chapter 1 described the reasons why this support is necessary. Current development techniques are complex and yield applications which lack portability, maintainability and reusability.

This dissertation proposed a presentation support platform to address these problems. Chapter 2 assessed what features this platform should provide. Firstly, the main types of multimedia applications were analysed. These can be summarised as presentational, compositional and collaborative. Each of these types subsume the complexities of the previous one. Presentational applications involve accessing data items, analysing or processing them and displaying them on a workstation screen. Compositional applications involve relating data items together in presentation, either by synchronisation, timelines or hypermedia links. Collaborative applications involve multiple users cooperating from different workstations, e.g. in a conference. This chapter also described the complexities of multimedia environments. Environments are composed of communication facilities, operating systems, system and application services. A wide range of these is emerging and thus heterogeneity is a problem. The conclusion of this chapter is that different environments should provide uniform facilities to application developers. Heterogeneity can be masked by authoring interactive presentations using generic abstractions. These must provide facilities to specify requirements regarding access to devices, information presentation and event handling. Services must be built to realise authored presentations at run-time. Using such facilities a wide range of applications can be constructed and used. In due course such facilities will feed into standards, but experimental work is required first to demonstrate the concepts.

Chapter 3 described related research work. Many elaborate authoring models have been invented but are often constrained by ignoring the requirements of applications in emerging environments. Some useful abstractions for building applications are also documented.

Chapter 4 illustrates an authoring model to realise the highlighted requirements. A data model allows distributed multimedia data to be referenced by name as data objects. These objects can be tailored by marking named events which will be notified at run-time and by annotating attributes. Objects can be grouped together as composite objects known as contexts. A script language is provided to specify interactive presentation of a context. This language allows configurations of presentation objects to be defined to tailor the presentation requirements of each data object. These presentation objects have methods which can be used to control them at run-time. Event-driven features can be built using event rules, which invoke methods on objects if one or a combination of events are detected. Facilities are also provided for synchronisation of configurations and for group invocation of presentation objects. A realised script creates an active context which can communicate with its controlling application at run-time through invocations, events and callbacks. Many such contexts can be part of the same application and can be communicated with as a group, allowing for collaboration.

Chapter 5 discussed the object support facilities which must be provided to realise data and presentation objects, and thus mask out environment complexity. Four layers provide this support. A platform database stores information on the facilities available in the platform and is used for checking the semantics of particular operations. A presentation database allows data objects and contexts to be created. Presentation objects are realised as active objects which have autonomous threads of control. They are concerned with performing a presentation function, handling invocations, monitoring for events and performing event notifications. Presentation and event functions can be implemented internally or using an existing service. Active presentation objects provide methods for presentation and quality control, connection to compatible objects and marking events to be monitored for. Factories, which exist in distributed factory sites, create presentation objects. The configuration service can activate and connect the appropriate presentation objects for a particular data object. It provides these facilities without the client having to know irrelevant specifics of factory and presentation object organisation.

Chapter 6 outlined higher level services which realise the functionality of active contexts. These services are divided into modules for implementation flexibility. At run-time configuration, event monitoring, event-driven invocation, synchronisation and communication with a controlling application must be managed. Before this an active context must be created and initialised. This process involves pre-processing and parsing a script and initialising various data structures. Event monitoring is realised using finite state automata for efficiency. Detected event expressions are notified to a firing module for event-driven invocation. An approach to synchronisation is discussed which can synchronise object configurations originating on different devices.

Chapter 7 illustrated how the objectives of this research have been satisfied. It has been possible to build examples of presentational, compositional and collaborative applications, using the authoring approach. The application development process has been simplified through the use of reusable generic abstractions. Developed applications are more elegant in structure, more maintainable and more portable than those developed using conventional techniques.

8.2 Towards the Future

The current IMP platform is only the first prototype and it is envisaged that greater functionality is possible using the principles it embodies [**Bacon93a**]. Some of the ideas for future development are listed in this section.

8.2.1 Data Modelling and Language Extensions

There are many envisaged extensions to the data modelling and script facilities. This section briefly describes some possible extensions using example scenarios.

At the data modelling level, the notion of attributes could be extended to support inter-object relationships. These named relationships directly reference other data objects stored in the presentation database. They are used to store relationships such as `is_text_about` or `is_the_related_audio_to`.

Although a design decision was made to avoid turning the IMP language into a full programming language, experience has illustrated that the ability to express more run-time dynamism within the script would be beneficial. One extension is the parametrisation of rules. An example is a rule of the form:

```
rule someone_clicked(person) =
        between(person.Clicked, person.Appears, person.Disappears) ->
  {
    person$Source.pause();
    person>>About$Display.show(....);
    person>>About$Source.play();
  }
```

This rule is a more flexible version of the `Identify_Scarlet` rule which can apply to anyone. It specifies that if an event of type `Clicked` is notified by any object after an `Appears` and before a `Disappears`, then pause that object and display text about it. An object matching the rule parameters is defined as `person` within the scope of the rule. The `>>` references an inter-object relationship. Thus every object which is applicable to this rule (i.e. video objects about people) should have an attribute `About` which references related text (as well as marked events `Clicked, Appears` and `Disappears`).

Another scenario which illustrates the need for more dynamic expression mechanisms is adding unanticipated users to a conference. It is desirable to write applications which flexibly add and remove any number of conferencees. The support for this type of application is already available in the object and presentation layers of the platform. Expression mechanisms need to be enhanced to capitalise on it.

It is also desirable to allow the specification of configurations involving presentation objects with more than one input and/or output.

8.2.2 Object Support Extensions

The following envisaged extensions will require enhancement of the object support facilities:

- *More intelligent event monitoring*
  As processor speeds increase more complex monitoring such as tracking moving objects through motion video are possible. An example is to monitor for the appearance of any 'red motorcycles' and track their movements through subsequent frames. Tools will be developed to allow such complicated monitoring scenarios to be specified.

- *Cooperation between distributed sites*
  Applications which access data from remote sites are a realistic possibility. This is due to the availability of high bandwidth inter-networks and the ubiquity of multimedia object repositories and continuous media devices such as cameras and microphones. A situation can be envisaged involving configuration services located at each site. These could cooperate in creating and connecting remote source objects with other local presentation objects using an appropriate data transfer protocol.

- *Knowledge of the individual*
  Support for the Laboratory's active badge system [**Harter94**] has recently been integrated into IMP[1]. Active badges allow network users to know where colleagues are within the building. Using the IMP data modelling facilities one can model *person* data objects, e.g. John, Scarlet or Jean. Events can be associated with these of types such as `EnterRoom`, `LeaveRoom` and `ClickBadge`. In scripts, these events can be used interchangeably with other event types. Presentation objects have been implemented to realise the system support aspects.

  Using a rule of a similar form to `IdentifyScarlet`, Jean can specify that if she leaves the room and Ken enters and she has not yet returned, then play him the message she has prepared for him.

```
rule video_greeting = and ( inbetween(Me.LeaveMyOffice,
                                      Me.EnterMyOffice),
                            Ken.EnterMyOffice)  ->
{
  /* Play message */
}
```

  A future requirement is to store more information in the platform database about the location of devices within a building. For example, what devices individuals have on their desks. Using this information one can determine what presentation and interaction capabilities they have. Given this facility in conjunction with the active badge support it is possible to specify instructions such as "when Andy is in his office give me audiovisual connectivity". When Andy returns to his office, if he has the capability and he agrees to the connection then a conference could be established. The configuration service could deduce what equipment the requesting user has and what equipment Andy has and could then connect them.

8.2.3 Emerging Applications

Virtual environments are an emerging class of application of which two types can be identified.

Firstly, scenario-based applications which use interaction-driven access to stored information to give a user the impression of being in a particular location and the ability to manipulate items they may find there. A project of this nature is being investigated in collaboration with the University of Cambridge Department of Anthropology.

[1]Work completed after January 1994

Secondly, virtual collaborative environments which enhance existing conferencing models. In such an environment, users can share information and communicate more naturally with other users.

In order to support such applications it is apparent that facilities to help application developers write the controlling part of the application are required. Such support must assist with the managing of information associated with particular states in a scenario, or interaction between users in the collaborative environment.

Appendix A

Compositional Applications

This appendix provides a more complete description of the compositional applications introduced in chapter 7.

A.1 Interactive Singing Monster

This section illustrates the context data and script used by the interactive singing monster application.

A.1.1 Context Data

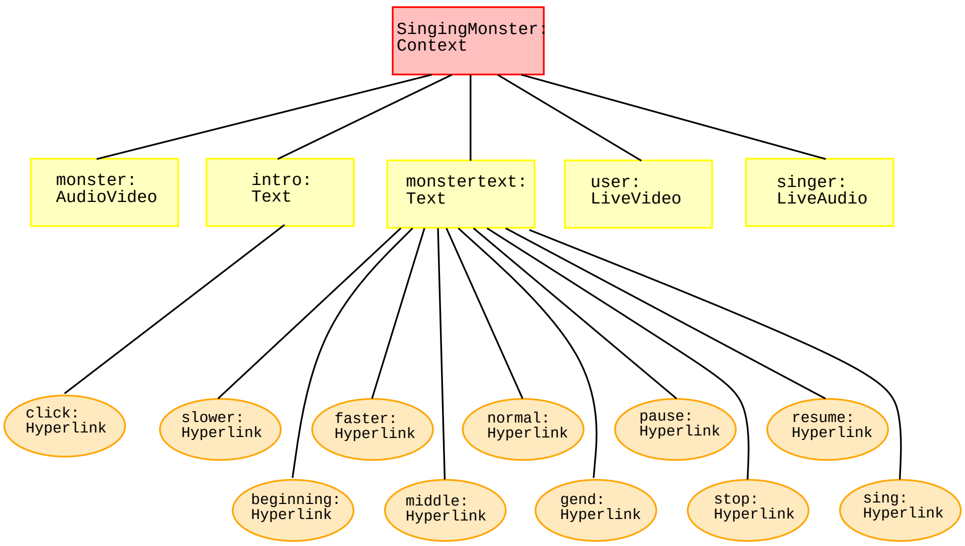


Figure A.1: Singing monster context data

A.1.2 Script

```
context SingingMonster;

external Start ;

monster | AVDisplay ;
intro | TextDisplay_Hyper ;
monstertext | TextDisplay_Hyper ;
user | VideoDisplay ;
singer | Speaker ;

set backingsinger = union(user, singer);
sync backingsinger;

rule Go = Start ->
                  {
                    intro$Display.show(80,300,250,50);
                    intro$Source.play();
                    monster$Display.show(80,20,200,200);
                  }


rule start_combing = intro.click ->
                  {
                    monster$Source.play();
                    intro$Display.hide();
                    monstertext$Display.show(20,250,350,100);
                    monstertext$Source.play();
                  }

rule stop_comb = monstertext.stop ->
                  {
                    monster$Source.stop();
                  }

rule slower_comb = monstertext.slower ->
                  {
                    monster$Source.slow(2);
                  }

rule faster_comb = monstertext.faster ->
                  {
                    monster$Source.fast(2);
                  }

rule normal_comb = monstertext.normal ->
                   {
```

```
                      monster$Source.normal_speed();
                    }

rule normal_comb = monstertext.normal ->
                    {
                      monster$Source.normal_speed();
                    }

rule pause_comb = monstertext.pause ->
                    {
                      monster$Source.pause();
                    }

rule resume_comb = monstertext.resume ->
                    {
                      monster$Source.resume();
                    }

rule beginning_comb = monstertext.beginning ->
                    {
                      monster$Source.goto_frame(0);
                    }


rule middle_comb = monstertext.middle ->
                    {
                      monster$Source.goto_frame(50);
                    }


rule end_comb = monstertext.gend ->
                    {
                      monster$Source.goto_frame(100);
                    }

rule sing_along = monstertext.sing ->
                    {
                      user$Display.show(10,20,60,60);
                      backingsinger$Source.play();
                    }


end;
```

A.2 Laboratory Presentation Example

This section illustrates the data and script necessary to realise the example referenced from the Laboratory presentation application. In this example the video section containing Scarlet loops so that the composite event may be experimented with.

A.2.1 Context Data

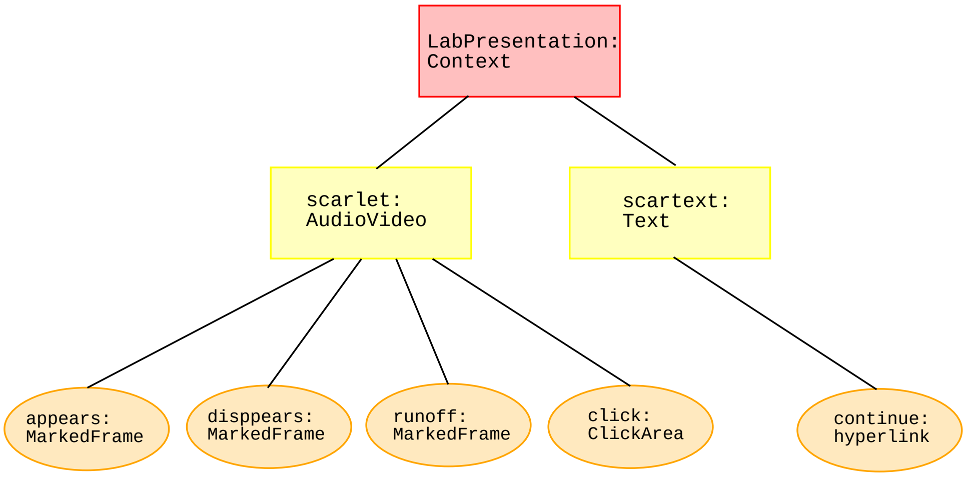


Figure A.2: Lab presentation context data

A.2.2 Script

```
context Lab Presentation;

external Start ;

scarlet | AvDisplay_Default ;
scartext | TextDisplay_Hyper ;


rule Go = Start ->
                    {
                      scarlet$Display.show(80,20,200,200);
                      scarlet$Source.play();
                     }


rule loop = scarlet.runoff ->
                    {
                      scarlet$Source.goto_frame(0);
                    }
```

```
rule in_position =
  and(
        inbetween(scarlet.appears,
                  scarlet.disappears),
      scarlet.click)

        ->
                     {
                      scarlet$Source.pause();
                      scartext$Display.show(80,270,250,80);
                      scartext$Source.play();
                    }


rule restart = scartext.continue ->
                    {
                      scartext$Display.hide();
                      scarlet$Source.resume();
                    }

end;
```

Appendix B

Collaborative Applications

This appendix provides a more complete description of the collaborative applications introduced in chapter 7. Relevant sections of controlling applications are included in a pseudo-code or pictorial notation.

B.1 IMP Phone

B.1.1 Context Data

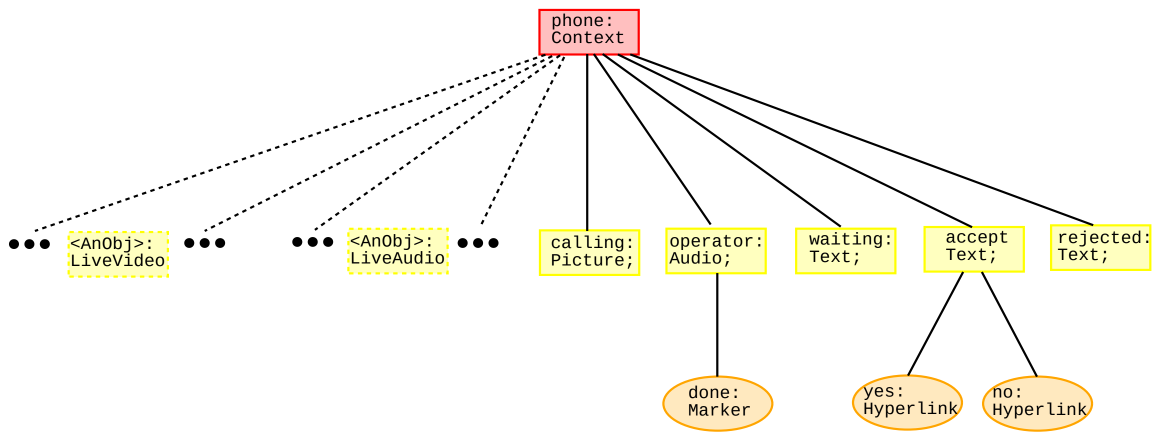


Figure B.1: IMP phone context data

B.1.2 Script

```
context phone(callerV, callerA, receiverV, receiverA);

external CStart, RStart;
external CConnect, CReject;

callerV | VideoDisplay;
CallerA | Speaker;
receiverV | VideoDisplay;
receiverA | Speaker;
calling | PictureDisplay;
operator | Speaker;
waiting | TextDisplay_Hyper;
accept | TextDisplay_Hyper;
rejected | TextDisplay_Hyper ;

set call = union(callerV,callerA);
set recv = union(receiverV,receiverA);
sync call;
sync recv;

rule Start_Caller = CStart ->
             {
               waiting$Display.show(20,240,260,30);
               waiting$Source.play();
              }


rule Start_Receiver = RStart ->
              {
               calling$Display.show(50,20,200,200);
               calling$Source.play();
               operator$Source.play();
               accept$Display.show(20,255,260,30);
               accept$Source.play();
              }

rule Connect = accept.yes ->
              {
               operator$Source.stop();
               calling$Display.hide();
               callerV$Display.show(20,20,260,260);
               callerV$Source.play();
              }

rule Reject = accept.no ->
              {
```

```
                  operator$Source.stop();
                  calling$Display.hide();
                 }

rule freeze = operator.done ->
                {
                  operator$Source.pause();
                }

rule reminder = delay(operator.done,5.0) ->
                {
                  operator$Source.goto_frame(0);
                  operator$Source.resume();
                }

rule receive_accept = accept.yes -> {}

rule receive_reject = accept.no -> {}

rule CShow = CConnect ->
                {
                  waiting$Display.hide();
                  receiverV$Display.show(20,20,260,260);
                  receiverV$Source.play();
                }

rule CHide = CReject ->
                {
                  waiting$Display.hide();
                  rejected$Display.show(20,20,260,260);
                  rejected$Source.play();
                }

end;
```

B.1.3 Controlling Application

```
--
-- IMP phone controlling application
--

VAR caller, receiver : Context ;


-- Acts on the decision of the receiver

PROCEDURE decide(cont: Context; rule: STRING; ch : Choice)
BEGIN
  IF (ch == ACCEPT) THEN

    -- Show the receiver to the caller

    AContextExternalEvent(caller, "CConnect");
  ELSE

    -- remove the receiver's workspace and inform the caller

    DestroyContext(caller);
    AContextExternalEvent(caller, "CReject");
END;


-- Indicate a command line error and exit

PROCEDURE ClError()
BEGIN
  fprintf(Stdout,"Usage : iphone [receiving workstation]\n");
  exit(0);
END;


-- Main program section
-- Takes parameters from command line
-- of workstation to contact

PROCEDURE Body(argc : INTEGER; argv: ARRAY [] OF STRING)

  VAR callV, recV, callA, recA : STRING;
BEGIN

  IF (argc != 2) THEN ClError();

  -- Initialise IMP library
```

```
  IMPinit();

  -- Assumes there is a database which stores
  -- names of workstations and associated AV devices

  IF (AVlookup(Localworkstation, callA, callV) < 0) THEN ClError();
  IF (AVlookup(argv[1], recA, recV) < 0) THEN ClError();

  -- Activate contexts for caller and receiver

  caller = ActivateContext("phone.imp", 4, (callV, callA, recV, recA));
  receiver = ActivateContext("phone.imp", 4, (callV, callA, recV, recA));

  -- Show the workspaces on the relevant screens

  AContextRealise(caller, Localworkstation, "Imp Phone", 300, 300);
  AContextRealise(receiver, argv[1], "Imp Phone", 300, 300);

  -- Add a callback so that this application knows if the receiver
  -- accepts or rejects the call

  AContextAddCallback(receiver, "Connect", decide, ACCEPT);
  AContextAddCallback(receiver, "Reject", decide, REJECT);

  -- Signal events to start both the caller and receiver

  AContextExternalEvent(caller, "CStart");
  AContextExternalEvent(receiver, "RStart");

END;
```

B.2 Opera Meeting

B.2.1 Context Data

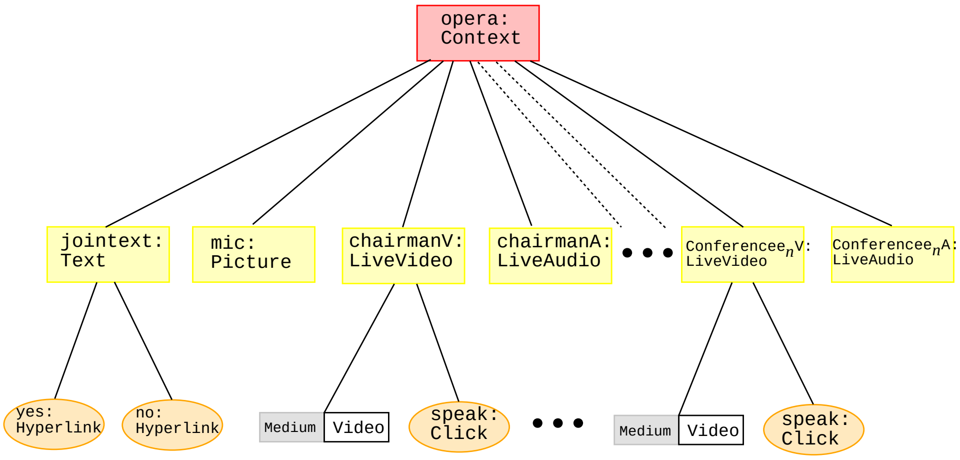


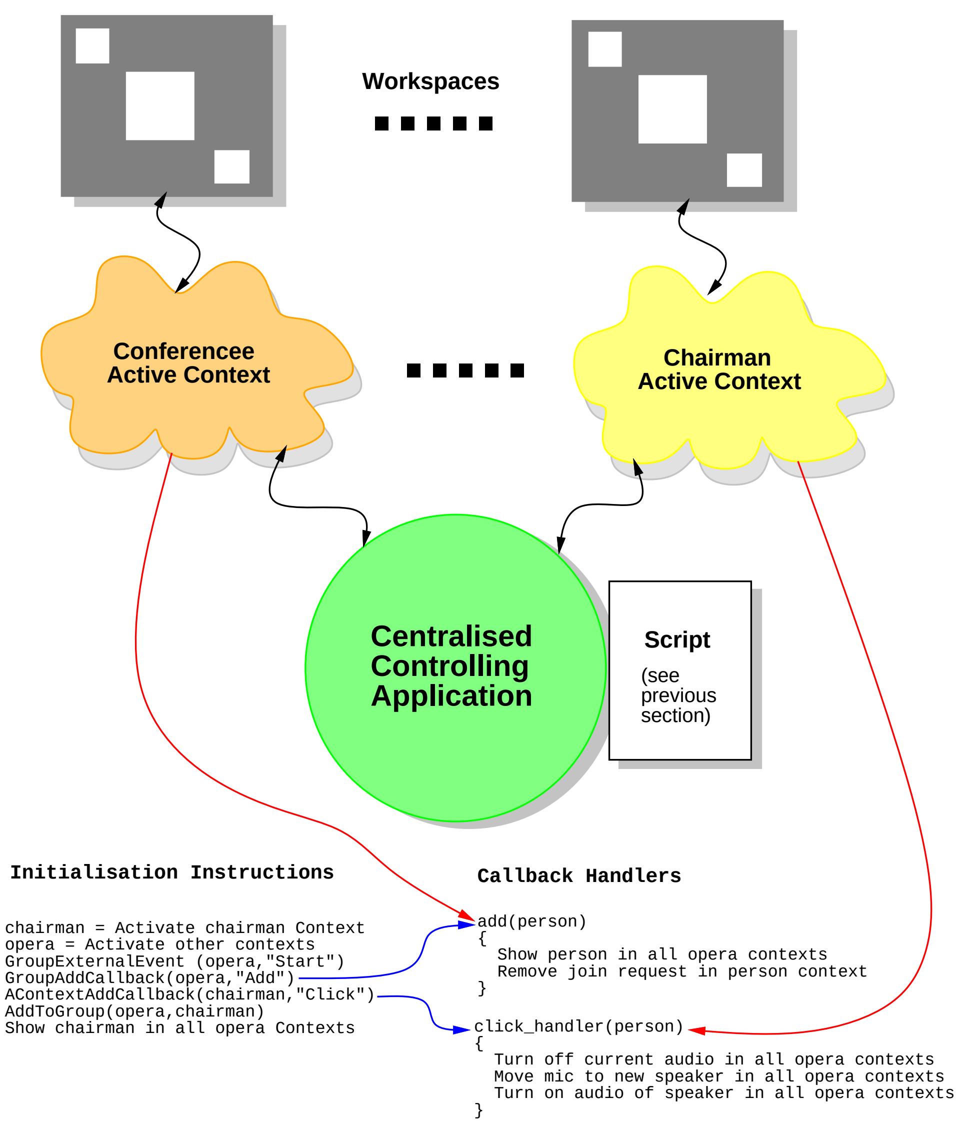
Figure B.2: Opera meeting context data

B.2.2 Script

```
context opera;

external Start;
external QoSEvent;

jointext | TextDisplay_Hyper ;
mic | PictureDisplay ;
chairmanV | VideoDisplay ;
chairmanA | Speaker ;
...
...
conferencee_nV | VideoDisplay ;
conferencee_nA | Speaker ;


set chair = union(chairmanV, chairmanA);
sync chair;
...
...

set vid = select(medium=video);

rule Go = Start -> {
                     jointext$Display.show(80,250,200,80);
                     jointext$Source.play();
                   }
```

```
rule add = jointext.yes -> {}

rule kill = jointext.no -> {}


rule chairmanVspeak = chairmanV.speak -> {}
...
...
rule conferenceeVspeak = conferenceeV.speak -> {}


rule degrade = QoSEvent ->
                 {
                   vid$Source.set_rate(10);
                   vid$Source.set_resolution(0);
                 }

end;
```

B.2.3 Controlling Application



Bibliography

[**Anderson91**] D. P. Anderson and G. Homsy. *Synchronisation Policies and Mechanisms in a Continuous Media I/O Server.* Technical Report UCB/CSD 91/617, Computer Science Division, University of California, Berkeley, February 1991. (p 93)

[**ANSA90**] Architecture Projects Management Limited, Poseiden House, Cambridge. *Advanced Networked Systems Architecture Testbench Implementation Manual*, 1990. (pp 3, 76, 82)

[**Bacon93a**] J. M. Bacon et al. *A Development Platform for Multimedia Applications.* Case for Support (Awarded as SERC grant ref: GR/J 42007), March 1993. (p 114)

[**Bacon93b**] J. M. Bacon et al. *Storage and Presentation support for Multimedia Applications in a Distributed, ATM Network Environment.* Technical Report 295, University of Cambridge Computer Lab, 1993. (p i)

[**Bates93**] J. Bates and A. R. Brooks. *Configurable Multimedia Processing Objects.* In Collected Abstracts from the Fourth International Conference on Network and Operating System Support for Digital Audio and Video. ACM, IEEE, IEE, 1993. (p i)

[**Bates94**] J. Bates and J. M. Bacon. *A Development Platform for Multimedia Applications in a Distributed, ATM Network Environment.* In International Conference on Multimedia Computing and Systems. IEEE, May 1994. (p i)

[**Bellcore93**] Information Networking Research Laboratory. Bellcore. *Touring Machine System.* Communications of the ACM, January 1993. (p 31)

[**Bertino88**] E. Bertino et al. *Query Processing in a Multimedia Document System.* ACM transactions on office information systems, 6(1):1–41, 1988. (p 21)

[**Blair91**] G. Blair et al., editors. *Object-Oriented Languages, Systems and Applications.* Pitman, 1991. (p 69)

[Blair92] G. Blair. *Distributed Multimedia Application Study.* Technical Report MPG-92-11, Department of Computing, Lancaster University, 1992. (p 6)

[Blair93] G. Blair and T. Rodden. *The Challenges of CSCW for Open Distributed Processing.* Technical Report MPG-93-24, Department of Computing, Lancaster University, 1993. (p 11)

[Blakowski91] G. Blakowski et al. *Tools for Specifying and Executing Synchronised Multimedia Presentations.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (p 93)

[Bly93] S. Bly et al. *Media Spaces: Bringing People Together in a Video, Audio and Computing Environment.* Communications of the ACM, January 1993. (p 10)

[Breiteneder92] C. Breiteneder et al. *Object Frameworks*, chapter Modelling of Audio/Video Data. University of Geneva, 1992. (p 24)

[Brooks94] A. R. Brooks. *Configuration of Distributed Multimedia Applications.* PhD dissertation, Lancaster University, 1994. In Preparation. (pp 32, 33)

[Brown89] H. Brown. *Standards for Structured Documents.* The Computer Journal, 32(6), 1989. (p 28)

[Bulterman91] D. Bulterman et al. *A Structure for Transportable Dynamic Multimedia Documents.* In Multimedia for now and the future, pages 137–155. USENIX, Summer 1991. (p 23)

[Bulterman93] D. Bulterman. *Specification and Support of Adaptable Networked Multimedia.* Multimedia Systems, 1(2):68–76, 1993. (p 23)

[Campbell93] A. Campbell et al. *Integrated Quality of Service for Multimedia Communications.* In Proceedings of INFOCOM '93, March 1993. Also appears as Lancaster University Department of Computing Technical Report. (pp 13, 82)

[Cardelli88] L. Cardelli. *Building User Interfaces by Direct Manipulation.* In SIGGRAPH, pages 156–166, 1988. (p 60)

[Christodoulakis86a] S. Christodoulakis et al. *Multimedia Document Presentation, Information Extraction and Document Formation in MINOS: A Model and a System.* ACM Transactions on Office Information Systems, 4(4):345–383, October 1986. (p 21)

[Christodoulakis86b] S. Christodoulakis et al. *The Multimedia Object Presentation Manager of MINOS: A Symmetric Approach.* In Proceedings of SIGMOD. ACM, 1986. (p 21)

[Conklin87] J. Conklin. *Hypertext: An Introduction and Survey.* IEEE Computer, pages 17–41, September 1987. (p 9)

[Cook90] R. Cook. *Desktop Video Studio.* BYTE, pages 229–237, February 1990. (p 8)

[Coulson91] G. Coulson et al. *Protocol Support for Distributed Multimedia Applications.* In Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991. (pp 32, 93)

[Coulson93] G. Coulson. *Multimedia Application Support in Open Distributed Systems.* PhD dissertation, Department of Computing, Lancaster University, 1993. (pp 32, 33, 74, 82)

[Crosby93] S. Crosby. *In-call Renegotiation of Traffic Paramters.* In Proceedings of INFOCOM '93, March 1993. (pp 13, 86)

[de Mey92] V. de Mey et al. *Object Frameworks*, chapter Visual Composition and Multimedia. University of Geneva, 1992. (pp 25, 26, 60, 74)

[Fish93] R. Fish et al. *Video Informal Communication.* Communications of the ACM, pages 48–61, January 1993. (pp 10, 32)

[Fujikawa91] K. Fujikawa et al. *Multimedia Presentation System "Harmony" with Temporal and Active Media.* In Multimedia for now and the future, pages 75–93. USENIX, Summer 1991. (pp 9, 21, 60)

[Gall91] D. L. Gall. *MPEG: A Video Compression Standard for Multimedia.* Communications of the ACM, 34(4), 1991. (p 12)

[Gehani92] N. H. Gehani et al. *Composite Event Specification in Active Databases: Model and Implementation.* In Proceedings of the 18th VLDB Conference, pages 327–338, 1992. (pp 51, 95)

[Gibbs92a] S. Gibbs et al. *Object Frameworks*, chapter Audio/Video Databases: An Object-Oriented Approach. University of Geneva, 1992. (p 25)

[Gibbs92b] S Gibbs et al. *Object Frameworks*, chapter A Programming Enviroment for Multimedia Applications. University of Geneva, 1992. (p 25)

[Gibbs93a] S. Gibbs et al. *Visual Objects*, chapter Data Modelling of Time-based Media. University of Geneva, 1993. (pp 24, 25)

[Gibbs93b] S. Gibbs et al. *Visual Objects*, chapter Application Construction and Component Design in an Object-Oriented Multimedia Framework. University of Geneva, 1993. (p 25)

[Greenberg] S. Greenberg et al. *Human and Technical Factors of Distributed Group Drawing Tools.* Interacting with Computers, 4(3):364–392. (p 10)

[Haake92] J. M. Haake and B. Wilson. *Supporting Collaborative Writing of Hyperdocuments in "SEPIA".* In CSCW, pages 138–156, 1992. (p 10)

[Harter94] A. Harter and A. Hopper. *A Distributed Location System for the Active Office*. IEEE Network, 8(1), 1994. (p 116)

[Hill93] R. D. Hill et al. *Rendezvous Language*. Communications of the ACM, pages 62–67, January 1993. (p 32)

[Hodges89] M. E. Hodges et al. *A Construction Set for Multimedia Applications*. IEEE Software, pages 37–43, January 1989. (pp 9, 18)

[Hodges91] M. E. Hodges and R. M. Sasnett. *Plastic Editors for Multimedia Documents*. In Multimedia for now and the future, pages 463–474. USENIX, Summer 1991. (pp 18, 19)

[Hopper86] A. Hopper and R. M. Needham. *The Cambridge Fast Ring Networking System (CFR)*. Technical Report 90, Computer Laboratory, University of Cambridge, June 1986. (p 13)

[Hopper90] A. Hopper. *Pandora - An Experimental System for Multimedia Applications*. ACM Operating Systems Review, 24(2), April 1990. Also Olivetti Research Technical Report. (pp 7, 13, 80)

[Hyden94] E. A. Hyden. *Operating System support for Quality of Service*. PhD dissertation, University of Cambridge Computer Laboratory, 1994. Submitted for examination. (pp 13, 86)

[ISO/IEC93a] ISO/IEC. *Coded Representation of Multimedia and Hypermedia Information Objects*. Draft standard, 1993. (pp 9, 28)

[ISO/IEC93b] ISO/IEC. *Hypermedia/Time-based Structuring Language*. Draft standard, 1993. (pp 9, 28)

[Johnson75] S. C. Johnson. *YACC- Yet Another Compiler-Compiler*. Technical Report 32, Bell Laboratories, Murray Hill, New Jersey, 1975. (p 103)

[Kim89] W. Kim and F. Lochovsky, editors. *Object-Oriented Concepts, Databases and Applications*, chapter Active Objects: Realities and Possibilities. ACM press, 1989. (p 69)

[Korth91] H. Korth and A. Silberschatz. *Database System Concepts*. McGraw-Hill, second edition, 1991. (p 68)

[Lesk75] M. E. Lesk and E. Schmidt. *Lex- a Lexical Analyzer Generator*. Technical Report 39, Bell Laboratories, Murray Hill, New Jersey, 1975. (p 103)

[Leslie92] I. M. Leslie et al. *Pegasus- Operating System Support for Distributed Multimedia Systems*. Operating Systems Review, pages 69–78, 1992. (pp 8, 13)

[Lo93] S. L. Lo. *A Modular and Extensible Network Storage Architecture*. PhD dissertation, University of Cambridge Computer Laboratory, 1993. (pp 14, 80)

[Mackay89] W. E. Mackay and G. Davenport. *Virtual Video Editing in Interactive Multimedia Applications.* Communications of the ACM, 32(7):802–810, July 1989. (pp 9, 18)

[Manna74] Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill, 1974. (p 95)

[Markey91] B. D. Markey. *Emerging Hypermedia Standards.* In Multimedia for now and the future, pages 59–74. USENIX, Summer 1991. (p 28)

[McAuley89] D. McAuley. *Protocol Design for High Speed Networks.* PhD dissertation, Computer Laboratory, University of Cambridge, September 1989. also as University of Cambridge Technical report 186. (pp 13, 86)

[Meyrowitz86] N. Meyrowitz. *Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework.* In Proceedings OOPSLA '86, pages 186–201. ACM, September 1986. (p 10)

[Moody93] K. Moody et al. *Storage, Programming and Display of Multimedia Objects.* In Future Trends in Distributed Systems. IEEE, 1993. Also appears as University of Cambridge Computer Laboratory Technical Report 294. (p i)

[Nicolaou90] C. Nicolaou. *A Distributed Architecture for Multimedia Communication Systems.* PhD dissertation, Computer Laboratory, University of Cambridge, December 1990. Also University of Cambridge Computer Laboratory Technical Report number 220. (pp 13, 82)

[Postel88] J. B. Postel et al. *An Experimental Multimedia Mail System.* ACM Transactions on Office Information Systems, 6(1):63–81, 1988. (p 9)

[Reynolds85] J. K. Reynolds et al. *The DARPA Experimental Multimedia Mail System.* IEEE Computer, pages 82–89, October 1985. (p 9)

[Scheifler86] R. W. Scheifler and Gettys J. *The X Window System.* ACM Transactions on Graphics, 5(2):79–109, April 1986. (p 3)

[Sreenan93] C. J. Sreenan. *Synchronisation Services for Digital Continuous Media.* PhD dissertation, University of Cambridge Computer Laboratory, 1993. Also as University of Cambridge Computer Laboratory Technical Report 292. (pp 14, 86, 93, 103)

[Sultana92] R. E. Sultana. *Storage Management for Multimedia Structured Objects.* Internal Report, University of Cambridge Computer Laboratory, November 1992. (p 84)

[Tang] J. C. Tang. *Finding from Observational Studies of Collaborating Work.* International Journal of Man-Machine Studies, 34:143–160. (p 10)

[Thomson90] S. E. Thomson. *A Storage Service for Structured Data*. PhD dissertation, University of Cambridge Computer Laboratory, 1990. (p 83)

[Vazirgiannis93] M. Vazirgiannis and C. Mourlas. *An Object-Oriented Model for Interactive Multimedia Presentations*. The Computer Journal, 36(1):78–86, 1993. (pp 26, 60)

[Wallace91] G. K. Wallace. *The JPEG Still Picture Compression Standard*. Communications of the ACM, 34(4), 1991. (p 12)

[Wilson88] I. D. Wilson. *Use of a Winchester Disc for Pandora*. Internal report, Olivetti Research Laboratory, Cambridge, December 1988. (p 14)

[Woelk86] D. Woelk et al. *An Object-Oriented Approach to Multimedia Databases*. In Proceedings SIGMOD '86, pages 311–325. ACM, June 1986. Published as SIGMOD Record 15(2). (p 20)

[Woelk87] D. Woelk and W. Kim. *Multimedia Information Management in an Object-Oriented Database System*. In Proceedings of the 13th VLDB Conference, pages 319–329, 1987. (p 20)

[Wray89] S. Wray. *Pandora Software Specification*. Technical Report, Olivetti Research Laboratory, Cambridge, March 1989. (pp 12, 86, 103)

[Wu93] Z. Wu. *A New Approach to Implementing Atomic Datatypes*. PhD dissertation, University of Cambridge Computer Laboratory, 1993. Submitted for examination. (p 68)