A Development Platform for Multimedia Applications in a Distributed, ATM Network Environment

John Bates and Jean Bacon
Computer Laboratory
University of Cambridge
Cambridge CB2 3QG, England

Abstract

Using current development techniques multimedia application creation is an ad hoc process yielding applications rooted in particular environments. We have designed authoring and presentation facilities which manage the creation and display of interactive multimedia presentations for applications in a distributed context. We illustrate with our prototype platform and applications developed using it.

1 Introduction

A multimedia environment integrates continuous media such as digital audio and video with the text and still pictures already supported in standard distributed computing environments. Workstations with bit-mapped displays and running windowing systems allow control over the presentation of visual media. Speakers allow audio to be played. Input devices such as cameras and microphones are becoming standard for the capture of audio and video. Storage services for continuous media have been developed and ATM networking technology supports the transport of multiservice traffic.

Using such technology a wide range of applications is possible. However, due to the complex nature of current environments, although application concepts are easy to formulate and express, implementing them using current program development techniques involves cumbersome and ad hoc manipulation of environment-specific devices. This leads to applications which are difficult to maintain and which are tailored towards particular environment configurations.

Our work addresses these problems by providing a platform of presentation support between the application developer and the environment (see figure 1). We call this platform IMP (Interactive Multimedia Presentation). It provides application developers with authoring tools to express their requirements in a manner which reflects the function and structure of their application rather than the environment on which it is based. To support this approach we have built several services which realise the interaction and presentation requirements at run-time.

To understand what features should be provided by a presentation support platform we first examined the requirements of current and emerging classes of multimedia applications. We then looked at the facilities available in current and emerging distributed multimedia environments. This analysis is described in section 2. We also link our research to related work.

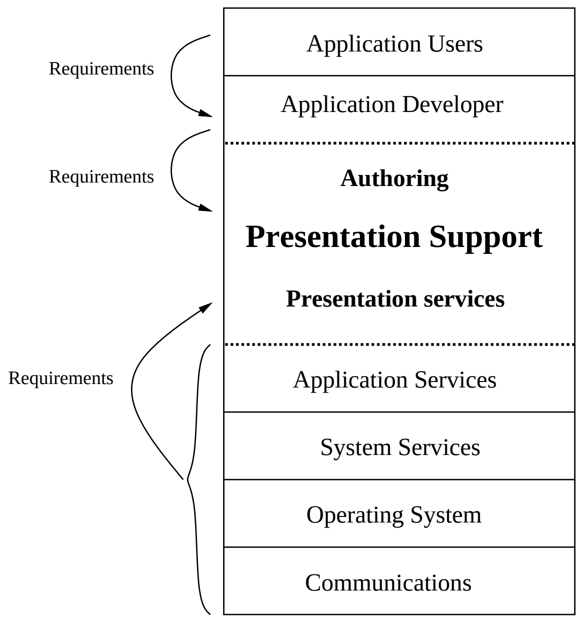

Figure 1: The role of presentation support

Our system's features for describing interaction and presentation requirements of applications are illustrated in section 3. In section 4 we describe how these are realised at run-time in terms of distributed services.

In section 5 we describe experimental applications built using our prototype platform. Section 6 describes how our work is proceeding and how we expect our platform to develop. Finally, section 7 summarises work to date and draws conclusions.

2 Platform support requirements

In this section we outline current and emerging multimedia applications and the environments in which they must run. From this analysis we extract the generic requirements of applications which involve interactive presentation of multimedia. This leads us to provide a standard set of abstractions. We also outline related work in various associated areas.

2.1 Multimedia applications

Application designers wish to make use of the rich facilities now available for interactive presentation.

Presentational applications can involve displaying multimedia data items, either live (e.g. audio and video of colleagues or television) or stored. They can also involve processing data items (e.g. image processing), analysing them (e.g. for image or speech recognition) or storing them.

Compositional applications relate several media components together and associate some control information. They thus subsume presentational concerns. Some *time-line* based applications involve associating timing control information so that certain media items are displayed at certain times. *Hypermedia* applications allow random access to media in response to user interaction, such as clicking while the mouse pointer is over selected display items. Such highly visually interactive applications are useful for Computer-Aided Learning (CAL).

Collaborative applications involve multiple cooperating users, such as in shared editing, drawing or audiovisual conferencing. They subsume compositional application concerns, for example when several people are involved in an audiovisual conference and a shared drawing, and they decide to use a pre-prepared presentation that all can see. These applications belong to the domain of Computer Supported Cooperative Working (CSCW).

2.2 Multimedia environments

Application developers must work with facilities available within environments. A series of standard environmental features are emerging. We will later illustrate with examples from the Cambridge multimedia environment.

The integration of continuous media has changed the view of the environment. Continuous media travels from source to sink as a stream of information. Such a stream has timing considerations, e.g. if it involves 25 frames per second then any processing and delivery of the stream must be done in a timely manner. The fact that streams take up resources for often undisclosed periods of time leads to the concept of Quality of Service (QoS) which must be supported by every component in a multimedia environment. If the resources of a device such as the network or a storage service are fully used, and more use is requested, then existing users must degrade their usage of the device by either destroying connections or degrading the quality of existing ones.

Developments to support continuous media include special hardware, ATM networks, protocols and services, special storage services and application services such as for synchronisation.

The environment is thus complicated by many different types of device, each with its own access semantics, and applications thus have a wide range of responsibilities.

2.3 Application responsibilities

We can summarise the added responsibilities of applications which include interactive presentation as

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

The application must manage connections to devices. Due to the constantly changing nature of multimedia environments there is no guarantee that devices of the same function will have the same interface.

By information presentation we mean accessing data from a devices, possibly processing or analysing it, displaying it in specific styles and possibly storing the result. The application can at any time change the nature of the presentation such as pausing the playing of video data or moving a text window. We can also require certain media components to synchronised, e.g. related audio and video.

By *events* we mean asynchronous occurrences at run-time. An application involving multimedia is not necessarily sequential. At any time, actions such as a user clicking on a person in a video or a word in a text document, a certain word of audio being spoken or a certain point in a video being reached, can mean that some response should be initiated. As well as these *user events* we also must consider *system events* such as (with some system configurations) the system informing us that we do not have enough bandwidth to maintain our level of QoS. By response to events we mean that the application must have policies to handle such pre-defined occurrences. In collaborative applications, an event from one user's context can affect other users'.

If all these responsibilities are realised on a per application basis then work is being replicated unnecessarily and the application is tailored to a specific environment.

2.4 Related work

Much interesting work has been done in trying to provide both authoring models for multimedia applications and programming abstractions with which they can be built. We are not aware of any work which integrates all application and environment issues.

Some of the authoring and presentation models have developed from hypermedia frameworks [3, 12].

Standards such as [8, 9] are intended to support the exchange of composite multimedia data and control information and do not take into account distribution.

Some approaches concentrate on building only one type of application such as [1] which provides an architecture for the construction of conferencing applications.

[5, 6] suggest a data model and a framework of active objects, configured graphically for application development.

[2] suggests stream, device and group abstractions for application construction.

In general the proposed authoring models put forward have been designed so that they run on some form of 'engine' and do not address the complexities of a real multimedia environment. Approaches using abstractions for presentation construction do not provide authoring methods to put these abstractions together.

We provide authoring and presentation facilities which manage the creation and display of interactive multimedia presentations for applications in a distributed context. Our approach is to provide data modelling and language facilities. We embody the principles of

- Reusability
- Generality
- Tailorability

Throughout our platform we make components general so that they can be reused in a wide range of contexts. We also allow specific requirements to be realised by making the system tailorable.

3 Authoring model

In this section we outline how the application developer can express the interactive presentation requirements of an application.

3.1 Data modelling

3.1.1 Data objects: We provide named data object abstractions for representing distributed data. We take into account the fact that distributed data can be of different media, different formats (encodings) and can have different devices as their sources. This data model makes no distinction between live and stored data.

We provide a database which can be given details of particular environments. Information about the types of device which are supported by the platform is registered with this database. The database also knows which types of media of which formats can have a source or sink on which devices.

When registering a new data item with the database it is necessary to describe details of media, source device and format. Once these details are registered the user never has to know anything about the object except its name and medium type, e.g. video, audio, text.

Thus a data object is a named encapsulation of a reference to a distributed data item. The database hides all details of system dependency. Only name and medium are required to use the object.

3.1.2 Data object instances: Application developers can create instances of multimedia data objects by

- marking events of interest
- annotating attributes.

An event of interest represents a named media dependent spatial or temporal portion of the object. Examples are a word of a text document, part of a picture or a specific frame of a video.

By marking events we are tailoring an object instance for a specific application. The aim of marking events is that during the presentation of a particular data object, if the event occurs then it is notified to the controlling service and action can be taken. Different events have different conditions- marking a certain word of a text document as a hypertextual event means that if the word is clicked on during presentation then the event is notified; with a marked frame of video, notification occurs if that particular frame is presented.

Attributes have no pre-defined semantics. They are annotated at the discretion of the user. They can, for example, define characteristics about the object such as a description of the contents, author or the creation date. Attribute names and their values can be used to query the database. Other uses will become apparent.

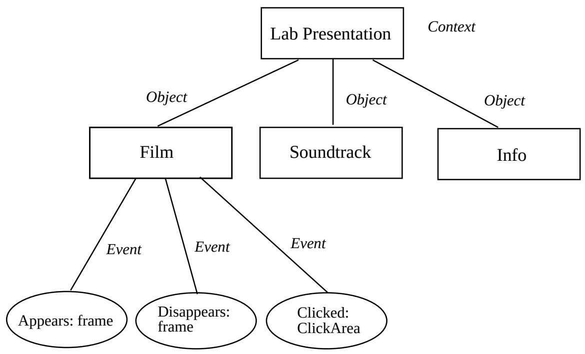

Figure 2: Part of the Lab Presentation context

3.1.3 Contexts: We allow data objects to be grouped together to form composite objects or *contexts*. These can represent objects related in certain ways, e.g. objects prepared for use in a certain application.

Imagine that we have created a context called "Lab presentation", illustrated in figure 2. Among its contents are a video object called "Film", an audio object called "Soundtrack" and a text object called "Info". Film has marked events "ScarletAppears" which signifies the frame at which our colleague Scarlet appears on screen, "ScarletDisappears" which signifies the frame at which she ceases to be on-screen and "Clicked" which indicates the part of the screen she occupies whilst on it. We will use this to build an example in the next section.

3.1.4 Manipulating the data model: The database has an object-oriented interface to promote an open authoring approach, i.e. tools can be built above it to manipulate the data model. Possible tools are

- *Graphical object modeller and browser*
  We have implemented a prototype tool to provide graphical manipulation of 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. They can be specialised by the marking of events and attributes. Queries can be used to retrieve desired classes of objects.

- *Graphical event marker*
  We envisage tools with which an application developer will be able to view particular data objects and, using direct manipulation, mark spatial and temporal events on them. An example is a tool which the user can use to play and pause video at particular frames and mark particular image details or frames.

3.2 Specifying interactive presentation

Using the above data model we can reference distributed data of different types using a uniform scheme. Using event marking we express our event notification requirements of an object with reference to interaction and presentation. We now propose a method of expressing the interactive presentation of these objects. We believe that standard programming language features are not suitable for expressing such requirements. We thus use a specially designed *script language* for this purpose.

3.2.1 Presentation configurations: We must be able to express how we want to process and display the data. To do this we define a configuration of presentation objects. A presentation object is an abstraction which performs some function on the data such as

- *Monitoring*
  Examining data for some phenomenon such as looking for someone's face in a video clip.

- *Filtering*
  Taking the data and changing it in some way

- *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 within which display components can be laid out.

- *Device access*
  Source and sink objects access data from or (in the case of a storage service) put data on a device.

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

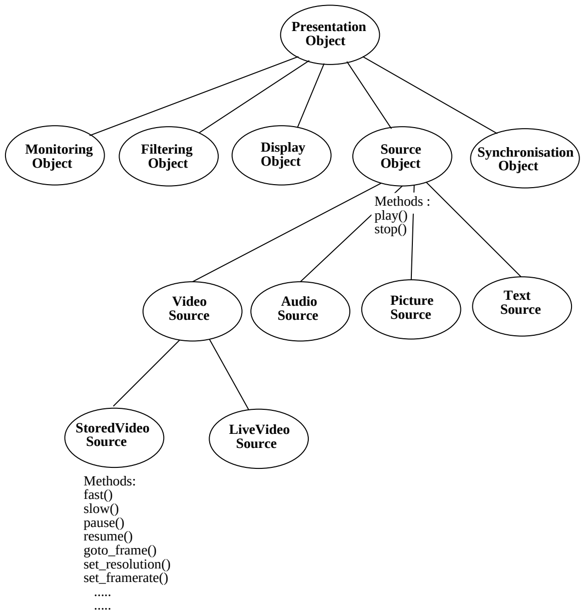

Figure 3: Part of a class hierarchy

Of the above types of presentation object, the first three types can be selected by the user in a script. The latter two types are inserted by the system "behind the scenes". More information about how these objects are implemented is given in section 4.4. For the moment we will discuss what is performed automatically and how we can use these abstractions for presenting our data.

The choice of abstractions is represented as a class hierarchy (see figure 3). Each presentation object has a class and associated methods. Methods can be added to the ones the object inherits from its parents as with the video object in the diagram.

A configuration is defined in the following way:

$\texttt{SourceObj | PresObj}_1\texttt{ |...PresObj}_n\texttt{ > SinkObj}$

As an example, imagine that we want to display the video data object "Film" using a video display object. This would require the following script statement:

```
Film | VideoDisplay ;
```

From information stored about data objects, the system can automatically create a source object to take data from the source data device.

Once we have set up this configuration we can use the presentation object methods to dynamically control the presentation process. In the above case, the object Film, being a video object, would have a source object with methods {`play`, `fast`, `slow`, `pause`, `resume`, `goto_frame`, `stop`, `set_framerate`, `set_resolution....`} and a video display object with methods {`show`, `move`, `resize`, `hide....`}. To access the methods, we use an invocation with the following parts:

- *The presentation configuration*
  This is named by the data object it processes

- *The presentation object(s)*
  This can be referenced by class or 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 one such object in particular.

- *The method to invoke*

- *The method parameters*

As an example, imagine we want to pause the video and move its window to a 50 by 50 area in the top left corner of our parent window. Invocations of the form

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

can be used. We reference the presentation objects by class and specify that the `pause` method should be invoked on all source objects in the Film configuration and the move and resize methods on all objects which are derived from the class display.

Notice that presentation objects have methods not only to affect the flow of information such as `pause` or `fast`, they also have methods to affect the quality of information presentation such as `set_resolution` or `set_framerate`. These are useful if we want to decrease the amount of resources used by particular objects. Lower resolution and frame rate for a video object implies less network bandwidth and fewer processor cycles to process and display.

3.2.2 Event-driven rules: Multimedia applications can 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. We have described how events can be marked on data objects. If these objects are used as part of a presentation at run-time then any occurrence of these marked events will be notified. We thus need a way of specifying what actions to take on the occurrence of events.

Event driven rules can be specified, of the form

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

- *Event expression*
  An event expression specifies one or a combination of events which will cause this rule to *fire*. Events can be those associated with objects or those specified to occur from outside the system- which can be defined in the script using the `external` declaration.

- *List of invocations*
  A series of invocations (of the form described above), on any of the objects in the context, are performed if the rule fires.

- *Rule name*
  The rule name represents an event which occurs when the rule fires. It can be incorporated into the event expressions of other rules.

As an example of an entire rule, we can specify that when the event ScarletDisappears associated with the object Film occurs, we want to go back to the beginning.

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

We can also combine primitive events into ordered combinations using certain operators. We have borrowed a general approach from the field of active databases [4] which is ideally suited to this form of application. We can express any temporal or spatial combination. For example we can specify with relation to object Film that 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 in_position =
  and(
    every(
     after(Film.ScarletAppears,
           or(Film.Clicked,
              Film.ScarletDisappears))),
    Film.Clicked)  ->
  {
    Film$Source.pause();
    Info$Display.show(80,270,250,80);
    Info$Source.play();
  }
```

This rule states that every time a ScarletAppears event occurs we know that some time later a Clicked or a ScarletDisappears event occurs. If this is the case, and it is true that the event is a Clicked event, then the event rule fires.

Other operators exist, such as `delay` which allows the integration of time offsets into event expressions.

3.2.3 Sets for group invocation: We allow data objects to be grouped into sets. 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. One emerging use for this is in policies for QoS. Imagine scenarios where the application must degrade its usage of network resources. 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".

Set operators are of the form {`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)));
```

which specifies all audio objects in the current context which are of Jean or Ken speaking. We can perform invocations on that set, such as

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

which will stop any which are playing.

3.2.4 Synchronising data presentation: We can use the `sync` declaration to specify that certain objects are to be synchronised in their playback. This is mostly used for lip-sync'ing between video and audio. We can define a set and then specify that it is to be played back in synchronisation-

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

3.2.5 Instantiation of script templates: We can define a script as an abstract template and then instantiate it with actual values when it is used. The first line of every script indicates which database context it is to be applied to. Imagine we had an application which implemented a video phone, i.e. you can see and hear your caller. The same script structure is required in many different situations. We can define a template, such as

```
context videophone(call_audio, call_video,
                   recv_audio, recv_video);
```

We can use the four parameter objects in the script to represent the four data objects to be used and then instantiate the script with the real values. If Jean wanted to call John then they could be `JeanAudio`, `JeanVideo`, `JohnAudio`, `JohnVideo`.

3.3 Controlling interactive presentation

We have now specified how multimedia data can be modelled and how interactive presentation can be specified using the modelled objects. It has been an overriding aim in this work that such presentations are not 'stand-alone'. Presentation support is required to help developers write the parts of their application which are difficult and unpleasant to do using current techniques. In this section we describe how a presentation is integrated with the rest of an application and how many such interactive presentations can be controlled from a single application, to support multi-user applications.

We provide a library which users can integrate with their programs. Its main functions are

- *Activation*
  To activate a script into what we term an *active context*, it is parsed with reference to a database context. If parsing is succesful the application program is returned a handle to the active context which it can use to perform other functions. An active context workspace can be configured to appear on different users' workstation screens.

- *External invocations and events*
  Using the context handle, application programs can affect the presentation dynamically. They can perform invocations on objects within the active context. They can also cause external events to be notified to the active context (these must have been expressed in the context using the `external` declaration).

- *Adding and handling callbacks*
  The application may want to be told when certain things happen within active contexts. It can add callbacks to event rules so that if they fire, the active context not only performs the invocations in the event rule but also informs the application that the rule has fired. The application can define special handler routines to manage such callbacks. Using external invocations and callbacks the application can perform functions not possible from the script such as dynamic layout of multi-user user-interfaces.

- *Context group management*
  Active context handles can be added to context groups. These are useful if an application involves the management of active contexts on different users' workstations. If, for example, we have an audiovisual conferencing application and a new user must be added then we want to show that new user appearing on every participant's display. Rather than performing this operation on every active context individually we provide all the above facilities of external invocation, external events and adding and handling callbacks for both single or groups of active contexts.

3.4 Authoring tools

Tools can be developed to assist the process of authoring. Examples are Graphical User Interface (GUI) builders to assist with display layout, tools to help novices use the authoring model or tools specialised for the development of particular types of application.

4 Presentation services

In this section we outline services which have been constructed to realise our authoring approach. We describe them in a top-down manner- in terms of what services we present our data modelling and interactive presentation requests to and how they turn them into the required functionality. We illustrate using our prototype IMP platform.

4.1 Supporting data modelling

A multimedia environment is likely to have a rich variety of devices thus yielding a large degree of heterogeneity. The IMP database provides both an application developer with data modelling facilities (section 3.1) and presentation services with the system-specific information required to present that data. The environment above which our prototype platform was constructed (figure 4) includes

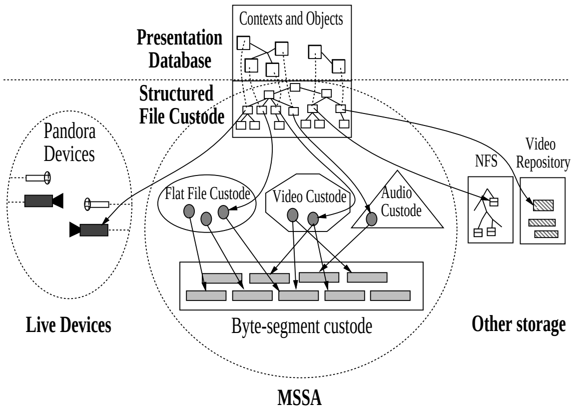

Figure 4: Experimental data environment

- *The traditional services* of a distributed environment are available- networked workstations using Ethernet and NFS
- *The Pandora system* [7] 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)* [10] uses *custodes* to manage individual data types. At the physical layer of the architecture reside byte segment custodes. At the logical layer reside file custodes for file classes which include traditional flat files, structured files and continuous medium files. Services have been built using a locally developed micro-kernel and ATM networks for data transport.

4.1.1 Presentation information: To build a database service we had to devise a way of representing data objects. This involved referencing distributed data and storing associated meta-data. We also needed an easy way of taking arbitary data objects and grouping them together as contexts.

We used the Stuctured File Custode (SFC) as it allows data to be represented in user-defined structures. It supports records and sequences of information fields, all of variable length. The SFC was hidden behind the object interface of the database (as shown in figure 4).

4.1.2 Platform information: To ensure legal platform operations within a particular environment, the following information must be stored:

- *Device templates*
  Information about what devices are supported by the system and what information has to be known about data which originates there. For example, for the MSSA we would need to store a storage service identifier (SSID) which uniquely identifies any object; for NFS data we would need to store a domain pathname and so on. When objects are added to the database and they are specified as originating on a particular device then the device template information must be provided.
- *Data object dependencies*
  Within the database are stored dependencies which specify which types of medium data can originate on which devices and support which formats. When adding objects, the database checks whether the medium supports the format and that the format is supported by the device.
- *Event templates*
  The database also stores which types of event exist in the system, their parameters and which media data types support them.
- *Presentation object class hierarchy*
  Information about available presentation objects is also stored. This is used by the configuration service (described below in section 4.5).

All of the above information may change, for example if we add support for new formats, new event types, new devices and so on. The more knowledge a database has about a particular environment, the more automatic services it can perform for the application developer during authoring and presentation.

4.2 The controlling application

An application which involves interactive presentation of modelled data is built up of its core language components plus IMP script(s). The IMP library contains functions to call a parser and create an active context. Active contexts can be self-moderating but some advanced applications may want more control of them via their external interface (see section 3.3). The library maintains internal data structures of active context references and any defined context groups. It contains calls to control single or groups of contexts. On the occurrence of callbacks, it must execute specified routines. Its implementation is fairly simple as all complexity is managed by a series of distributed services. This means that only high level management functions are performed from the controlling application.

4.3 Active contexts

An active context is an object instantiated when a script is applied to a context in the database. It initialises its state with a series of instructions to control presentation at run-time. It also creates and configures distributed objects and services to control the environment and manage aspects of the presentation.

We can sub-divide these stages into

- *Parsing*
  The script is first parsed for syntactic and semantic validity. Semantic validity is ensured by checking that every referenced data object, event, presentation object class and object method exists.

- *Creation of presentation object configurations*
  Device access, data processing and display are controlled by configurations of active presentation objects. The nature of these objects is described below in section 4.4. The process of creating, checking and implementing the configuration of these objects is performed by a configuration service. This is described in section 4.5.

- *Setting up event rules*
  The event expression part of the event rule is monitored for by an *event monitoring service* described in 4.6. The rest of the rule is put into an event rule structure indexed by name. This contains an *invocation list* which is a list of invocations to perform if an event expression is detected, and a *callback list* which is a list of application callbacks for use in the same circumstance.

- *Maintaining information on sets of objects*
  Details of named sets of objects are maintained in the state.

- *Synchronising*
  A synchronisation service is outlined in section 4.7.

The active context also has an external interface-accessible via the interface reference it returns as its handle to the controlling application. Using this, an application can notify external events, perform external invocations and add callbacks.

4.4 Presenting multimedia data

Presentation objects are implemented as active objects- objects in the object-oriented sense of having data and methods as well as having their own asynchronous threads of control. This means that, as well as being activated on receipt of a message, they can also perform asynchronous processing and send messages.

Presentation objects have methods to perform the following functions:

- *Presentation control*
  Processing 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....`}

- *Event marking*
  These methods are used by the system to tell the presentation object of events to monitor for. For example a synchronisation object monitors for video frame boundaries- but it has to be told which ones are of interest to an application.

- *Interconnection*
  These methods are used by the system to connect presentation objects together in configurations. This can be via port abstractions or through some form of mutually agreeable shared memory arrangement.

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.

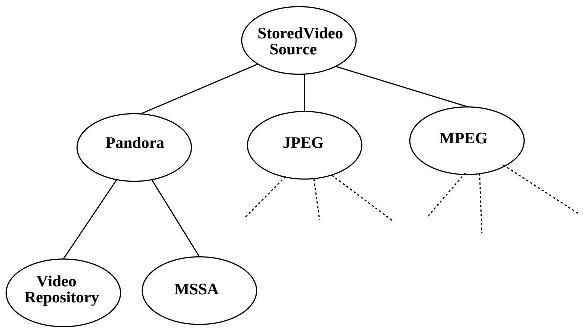

Figure 5: Continuation of a class hierarchy

4.5 Configuration service

The configuration service sets up script-specified presentation object configurations. It uses type and object information from the database. Its duties are

- *World creation*
  Factories create one or more classes of active presentation objects. They exist in *sites* which can be distributed. The configuration service obtains references to relevant factories (a *world*) for each presentation. Some factories need to be created per presentation, such as a display factory and associated workspace.

- *Type checking*
  Obviously incompatible objects cannot be linked together. For example we cannot say

  ```
  VideoSourceObject | AudioPresObject
  ```

  Therefore the configuration service will check that the output type of the preceding object is compatible with the input type of the next object.

- *Appropriate creation of presentation objects*
  Although to the author it appears that presentation objects are simply abstractions designed to process certain types of medium, supporting such abstractions is more complex due to the inherent heterogeneity of multimedia environments. Consider our earlier example of a video data object "Film" which is linked to a display object. Film is stored on a device and has a particular format. To the system, the class hierarchy of figure 3 continues in figure 5. We can see that for every abstraction, the system must provide components for all possible device and format combinations. Each has the same methods as the parent

abstraction to keep these system-dependent details transparent from the application developer. In the case of Film, if it originates on the MSSA and is of type JPEG then its source object is of class `Source_Video_Stored_JPEG_MSSA`. The system must use information stored in the database to work this out and select the correct class of presentation object.

- *Automatic interconnection*
  Having created the objects, the configuration service uses database information to determine the methods of connecting together the specific classes of object in the required configuration. It then performs this interconnection.

- *Automatic event marking*
  The configuration service informs presentation objects of events they must monitor for. It uses the event information associated with data objects and database presentation object class information to decide which marked events are relevant to be monitored for by which presentation object.

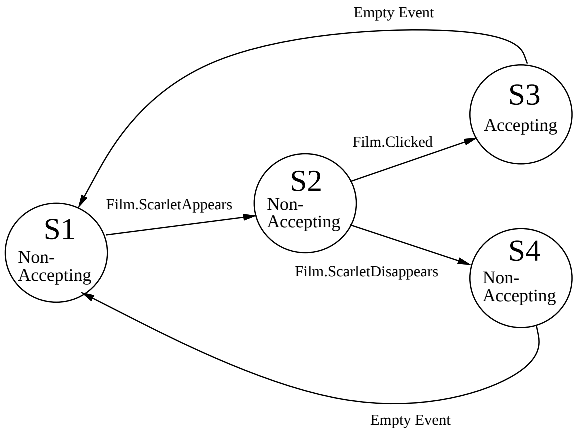

Figure 6: Example finite state machine

4.6 Event monitoring service

As described above, the invocations and application callbacks associated with the firing of an event rule are stored in lists which are accessed by the event rule's name. The event expression is registered with an event monitoring service which monitors for the occurrence of its individual events and updates its state accordingly. If the entire expression is detected, it will notify the keeper of the lists which will activate them.

Monitoring for event expressions is achieved by constructing finite state machines. Every time an event occurs, the machine changes state. States are either accepting, which indicate the expression has been detected, or non-accepting.

A finite state machine representing the expression in section 3.2.2 is represented in figure 6.

4.7 Synchronisation service

In our model a synchronisation presentation object is added by the system to every configuration of presentation objects involving continuous media. This provides jitter removal through buffering. A synchronisation service is informed of objects which require mutual synchronisation, such as related audio and video (lip-sync). A prototype synchronisation service has been constructed in our Laboratory [11]. In this model, synchronisation presentation objects are related together by a synchronisation group agent. This agent views logical time values from each synchronisation presentation object and adjusts their presentation rate to keep them in sync.

5 Experimentation

We have built experimental applications from all classes described in section 2.1, including

- *Interactive singing Monster*
  This application shows how objects can be configured and events can be used to control presentation. Using such techniques compositional applications can be supported. The application displays video and audio of a singing Monster which can be controlled interactively via a hypermedia style interface. The user can even sing along with the Monster if he/she so desires. This is achieved by displaying locally captured video in a window next to the monster and playing local audio.

- *Laboratory presentation*
  This application illustrates further concepts of how interaction can be based around activities in a presentation. The application presents views around our Laboratory. By interaction with the system certain information can be called up. The example associated with figure 2 was taken from this application.

- *Conferencing*
  We have built some prototype applications which allow multiple cooperating users to affect one another's context. One application allows our weekly "Opera group" meetings to take place without leaving our offices. Users are textually queried whether they wish to participate. If they agree then live video and audio from them is added to the workspaces of all participating users. This application also involves adjusting the quality of media presentation in response to resource availability.

6 Towards the future

Our current system is only the first prototype and we envisage that much more functionality is possible using the principles it embodies. Some of our ideas for future development are

- *Integration with emerging QoS concepts*
  Emerging multimedia applications with real users

are a strain on real environments. As methods emerge to specify what processors and networks can do in the event of over-use we would like to make these available at an authoring level to application developers.

- *More intelligent event monitoring*
  As processor speeds increase more complex monitoring such as tracking moving objects through motion video are possible. We will be developing tools and methods of marking such events on data objects.

- *Knowledge of the individual*
  We want to integrate support for our Laboratory's active badge system at the data modelling level. Active badges allow us to know where colleagues are within our building. We also want to store more information about what particular devices are where in a building, for example what devices individuals have on their desks, so that using this information one can determine what presentation and interaction capabilities they have. Using such an approach one can 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 we can talk. The configuration service could deduce what equipment I have, what he has, and could link us.

- *Extended facilities for collaboration*
  Many emerging applications involve collaboration. We want to be able to support applications which enter users into virtual collaboration environments, where they can share information and communicate.

7 Summary and conclusions

We have illustrated our work on authoring and presentation of multimedia for interactive applications in a distributed context. It is motivated by the inadequacies of current application development techniques.

The authoring model provides features necessary to enable the construction of a wide range of applications. A data model allows distributed data to be referenced as data objects. These can be tailored with events and attributes, and related as contexts. Interactive presentation of a context can be specified using a script language. Configurations of presentation objects can be used to tailor presentation requirements. Event-driven rules allow one or a combination of user and system occurrences to be handled. Sets of objects can be created for group invocation or synchronisation. Scripts are instantiated as active contexts. Several of these can interact with a controlling application to support cooperative applications.

Presentation services are required to coordinate an environment in the realisation of authored presentations. A database provides both users with data modelling facilities and presentation services with access to system-specific details. During a presentation, a script is parsed and used to initialise various data structures. The system-specific concerns of presentation objects are realised as active objects. A configuration service provides facilities for configuring these objects. An event monitoring service can detect complex event expressions. A synchronisation service can synchronise the presentation of related configurations.

We have constructed a prototype platform within the Cambridge multimedia environment. Experiments show that a range of applications can be built quickly and elegantly.

References

[1] Bellcore Information Networking Research Laboratory, "Touring Machine System", *Communications of the ACM*, January, 1993.

[2] G. Coulson, "Multimedia Application Support in Open Distributed Systems" *Ph.D. thesis*, Computing Department, Lancaster University, 1993.

[3] K. Fujikawa et al., "Multimedia Presentation System 'Harmony' with Temporal and Active Media", *Multimedia for now and the future*, USENIX, pp. 1-100, 1991.

[4] N. H. Gehani et al., "Composite Event Specification in Active Databases- Model and Implementation", *Proceedings of the 18th VLDB Conference*, pp. 327-338, 1992.

[5] S. Gibbs et al., "Data Modelling of Time-based Media", *Visual Objects*, University of Geneva, 1993.

[6] S. Gibbs et al., "Application Construction and Component Design in an Object-Oriented Multimedia Framework", *Visual Objects*, University of Geneva, 1993.

[7] A. Hopper, "Pandora - An Experimental System for Multimedia Applications", *ACM Operating Systems Review*, Vol. 24(2), April, 1990.

[8] ISO/IEC, "Hypermedia/Time-based structuring language", *Draft standard*, 1993.

[9] ISO/IEC, "Coded Representation of Multimedia and Hypermedia Information Objects", *Draft standard*, 1993.

[10] S. L. Lo, "A Modular and Extensible Network Storage Architecture", *Ph.D. thesis (and technical report #326)*, Computer Laboratory, University of Cambridge, 1993.

[11] C. J. Sreenan, "Synchronisation Services for Digital Continuous Media", *Ph.D. thesis (and technical report #292)*, Computer Laboratory, University of Cambridge, 1993.

[12] M. Vazirgiannis and C. Mourlas, "An Object-Oriented Model for Interactive Multimedia Presentations", *The Computer Journal*, Vol. 36(1) 1993.