Technical Report

UCAM-CL-TR-294
ISSN 1476-2986

Number 294

OPERA

Storage, programming and display of multimedia objects

Ken Moody, Jean Bacon, Noha Adly, Mohamad Afshar, John Bates, Huang Feng, Richard Hayton, Sai Lai Lo, Scarlet Schwiderski, Robert Sultana, Zhixue Wu

April 1993

ISSN 1476-2986

OPERA
Storage, Programming and Display of Multimedia Objects

Ken Moody, Jean Bacon, Noha Adly, Mohammed Afshar, John Bates, Huang Feng, Richard Hayton, Sai Lai Lo, Scarlet Schwiderski, Robert Sultana, ZhixueWu.

University of Cambridge Computer Laboratory

Abstract

This project aims to support the interactive display of synchronised multiple media types in workstation windows. This style of application needs high speed ATM networks and suitable protocols and operating systems; an infrastructure which exists at the University of Cambridge Computer Laboratory. Above this infrastructure we have designed and are building storage services (MSSA), a platform to support the creation and display of multimedia presentations (IMP) and a persistent programming language (PC++), for reliable and convenient programming of multimedia applications. This paper gives an overview of the work of the OPERA project in these three areas.

1. Introduction

We are building a display platform for multimedia applications above a multi-service storage architecture (MSSA). This style of application needs high speed ATM networks and suitable protocols and operating systems. Our work has built on such an infrastructure, see [Bacon 93, Chapter 22], to provide storage and presentation support for multimedia applications.

Current storage services are unable to meet the requirements of emerging application areas. There are both architectural and engineering reasons for this. A computing environment which supports multimedia (such as real-time video and audio) requires very large amounts of storage, the ability to express complex relationships between stored objects and synchronised delivery of media streams at guaranteed rates. A multi-service storage architecture (MSSA) has been designed for these, as well as traditional, applications [Bacon et al., 91].

An open storage architecture gives flexibility and extensibility. Conventional files, audio, video and structured objects are supported within a common architectural framework in MSSA and composite objects, such as a display representation, may have components of any of these storage types. The two-level hierarchy of servers in MSSA provides storage media and a byte segment abstraction at the low level and a variety of abstractions at the high level. Quality of service guarantees, which are essential for continuous media file types, are supported by *sessions* and *tickets*. These are arranged via the high level servers and used directly with the low level servers, thus avoiding the inefficiency that might be caused by the layered structure.

A platform for the creation and interactive display of multimedia presentations (IMP) is being developed. A script language allows a multimedia presentation to be specified in terms of objects, the relationships between them and the (composite) events that drive it. Presentation data is stored on a structured data service within MSSA (see Section 2) and component objects are stored on appropriate servers, and accepted and retrieved at guaranteed rates using session and ticket mechanisms. A presentation associated with a multimedia application is achieved by applying a script to the data representing the presentation to create a display.

We have developed a persistent programming language PC++, an extension to C++, for multimedia application programming [Wu et al. 93]. PC++ uses the MSSA for persistent storage and programmers may manipulate data structures stored across the special purpose servers in the distributed environment of MSSA. In PC++ it is possible to express complex relationships between objects and to make related changes to objects within a transaction. Transactions also give an abstraction for handling the independent failure modes of the components of distributed multimedia objects. PC++ uses an optimistic method of concurrency control which is appropriate for a multimedia environment in which conflict is rare, real time (quality of service) requirements must be met and continued working after failure is desirable.

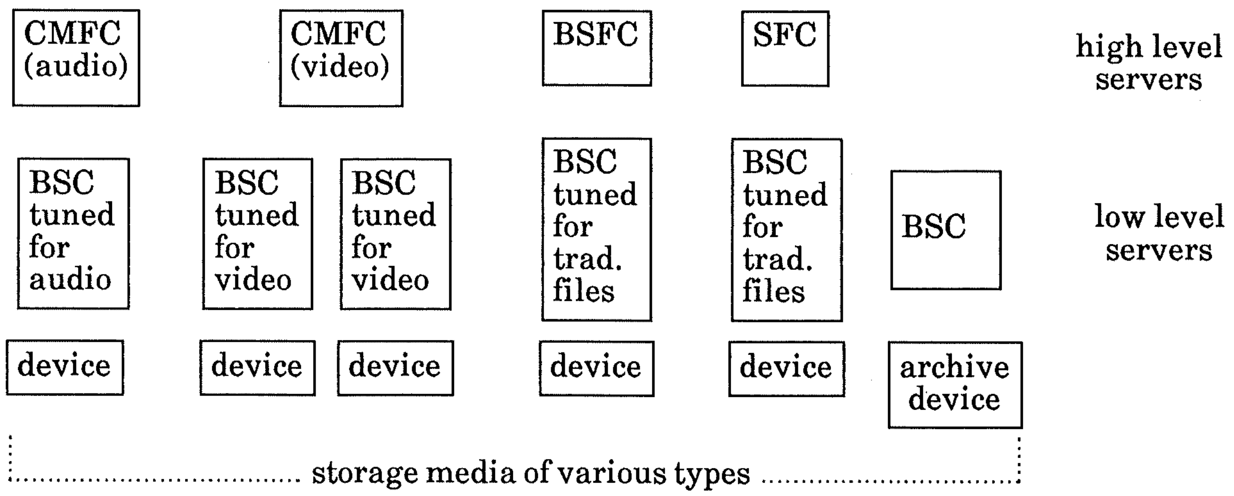
This paper gives an overview of the OPERA project. Storage is discussed in Section 2, presentation support in Section 3 and programming in Section 4. Section 5 outlines the current prototype implementations and Section 6 gives interim conclusions arising from this ongoing research. Related technical reports describe storage and presentation (TR 295) and PC++ (TR 296) in more detail.

2. Storage

Our storage architecture comprises an open, two-level hierarchy of servers. Details are given in [Bacon et al. 91, Lo 93].

- The servers at the low level, the byte segment custodes (BSCs), manage storage media of any type, support a common byte segment abstraction and provide quality of service guarantees for acceptance and delivery of data. The architecture is such that the special purpose servers at the high level, for example, those for video, audio, structured objects or conventional files, all employ the low level BSCs for storage.
- The servers at the high level provide storage abstractions for various types of stored object; for example, we have a byte stream file custode (BSFC) for conventional files, a structured file custode (SFC) and a continuous media file custode (CMFC). There may be multiple instances of each type of custode.
- A server at the low level (a BSC) can be involved directly with data transfers from or to a client (a workstation window, for example). Such transfers take place within a session which is organised through one of the high level custodes. The client acquires a number of tickets for use with the appropriate BSCs to achieve delivery of the files needed in a display.

Figure 1 shows an example of an MSSA configuration of servers.



high level servers		low level servers	
BSFC	= byte stream file custode	BSC	= byte segment custode
SFC	= structured file custode		
CMFC	= continuous medium file custode		

Figure 1 An example of an MSSA configuration.

Every object in the MSSA is named and protected by a storage service identifier (SSID) which is a capability labelled with a principal [Gong 89]. An SSID can be used to identify an object uniquely system-wide, and only the principal may use the SSID to access the object.

Location of objects in the distributed system is managed through container abstractions at two levels. A given logical container is managed by a specific file custode. A given physical container is managed by a specific byte segment custode.

2.1 The byte segment custodes (BSCs)

BSCs manage a variety of storage media which might range from non-volatile RAM to a terabyte optical jukebox. They present a common byte segment abstraction to the higher level custodes which may be used in any way that is appropriate. This also gives extensibility in that new media may be added within the architecture. An example of how the low level might be used is that a video file may be striped across a number of devices as a number of byte segments in a number of containers managed by different BSCs. A traditional, byte sequence file is likely to map onto a single byte segment. The fact that the low level imposes no specific structure means that arbitrary standard structures, such as MPEG for video or interleaved video and audio, can be supported at the high level.

We use extent-based storage allocation within a byte segment. A storage-block size may be associated with a byte segment, as a hint to the BSC, so that different media may be treated appropriately. We use NVRAM to store data and metadata in the implementation of the BSCs. This allows a fast acknowledgement that a secure write has taken place without the need for synchronous disk access. It also provides a convenient basis for implementing atomic write operations within the storage service and supporting application-level transactions.

2.2 Quality of service for media storage

If an object of a continuous medium type such as video is to be shown as part of a display then a certain rate of delivery must be guaranteed (at some probability) between the custode containing the stored object and the workstation. The mechanism we use to support this is the *session* which is controlled by a *ticket* which guarantees a certain rate of delivery for a specified time. A server can ensure that it does not issue more tickets than it can honour.

We take the view that the storage service should take care only of storage and its responsibility ends when the data reaches the client-end. The purpose of having sessions and tickets is to ensure that continuous media data are delivered in a way predictable by the client. For instance, we have to take account of the fact that data may be variable instead of fixed rate and might be lost in transmission [Lo 93].

2.3 Structured objects

We represent structured objects on *structured file custodes* (SFCs) [Thomson 90]. An SFC is lightweight compared with a full typed-object manager. It supports only byte sequences and storage service identifiers (SSIDs) as primitive storage types but the constructors *sequence*, *record* and *union* are used to create tree structures of arbitrary depth. Since the SFC can locate SSIDs within structured objects it can provide an **existence control** service by traversing them. The representation of structure means that locks can be associated with components of stored objects.

This simple data object store has been used to represent complex, multi-object presentations. An active multimedia database has been created on the SFC to support event driven display, see below.

2.4 Continuous media objects

Clients will use the continuous media file abstraction to store all data that is continuous in nature, whether it is audio, video or interleaved audio and video and whether it is encoded in MPEG, H.121 or any proprietary format. We may have multiple instances of the continuous media file custode (CMFC) running at the same time.

We can claim that the CMFC supports multiple formats because it enforces none. It is up to the client at the receiving end to decide how to handle a given stream. The stub at the client end is a "translator". It interprets the byte stream, identifies frame boundaries and extracts timing information. It then sends the data to other processing elements down the pipeline from server to user. For example, in the MPEG standard, the role of this translator is the "Medium Specific Decoder" (which is not part of the standard). Its role is to translate from the digital storage format to an ISO-11172 stream which is fed into an MPEG codec. Different data with different encoding schemes will have different translators.

3. Composition and display of multimedia presentations

The Pandora project [Hopper 90] gave us insight into the requirements of multimedia applications and created some prototype applications, such as video mail, which are in everyday use at the Laboratory. Additional support is needed if users are to create and interact with multimedia applications as opposed to just using those provided. Our development platform will allow multimedia applications to achieve their presentation requirements without the cumbersome, ad hoc engineering of the lower levels which is necessary in current systems.

- A user must be able to compose a multimedia presentation, as part of an application, and express requirements and policies for its dynamic, interactive display.

  The IMP (Interactive Multimedia Presentation) system [Bates 93] provides a general and flexible interface for use by individuals and groups. Two aspects are involved: the creation of structured **presentation data** and the expression of a **script** to drive a presentation.

- To support the creation of presentation data and meet the requirements expressed in the script we use the MSSA to provide **storage** of presentation structures and their components of various media types and **delivery** of these components at the right rate and at the right time (triggered by possibly composite events). In addition, we require **synchronisation** between components.

Figure 2 gives an overview of the components of the IMP system which are involved in creating presentation data and achieving a display. [Bacon et al. 93] gives details of how presentation objects are created and how a dynamic, interactive display is achieved.

We have built a *presentation object store* on the SFC (see Section 2.3). All objects which may be presented are registered here. An application may set up a number of *presentation objects* of registered media classes and tailor them to its own requirements by adding attributes and named events. A presentation object includes the SSID of the corresponding media object which is stored on the appropriate MSSA server. The various presentation objects are composed into a structured object which represents an entire presentation (the *presentation context* object), see Figure 2. The SSIDs of the component objects of various media types are therefore contained in this object and the events which drive an interactive presentation are named and marked in it.

The figure shows the IMP script through which the presentation is controlled dynamically. A script, written in the IMP language, comprises an imperative framework for a display together with declarative rules which indicate the actions to be taken when certain event expressions become true, see [Bacon et al. 93]. The events of interest to a given presentation, which are expressed in its script, are registered with a "presentation and monitoring service" and their occurrence during the presentation is notified to this sevice. When an event occurs (or an event expression becomes true) this service interacts with the presentation object store, which we have implemented as an **active** database. Objects may thus be retrieved from storage and delivered to the workstation, triggered by the firing of the corresponding events.

Finally, we use a network-based synchronisation service [Sreenan 92], which is independent both of the MSSA and the presentation world, to perform functions such as jitter removal, monitoring for frame events and achieving, for example, lip synchronisation between voice and video.

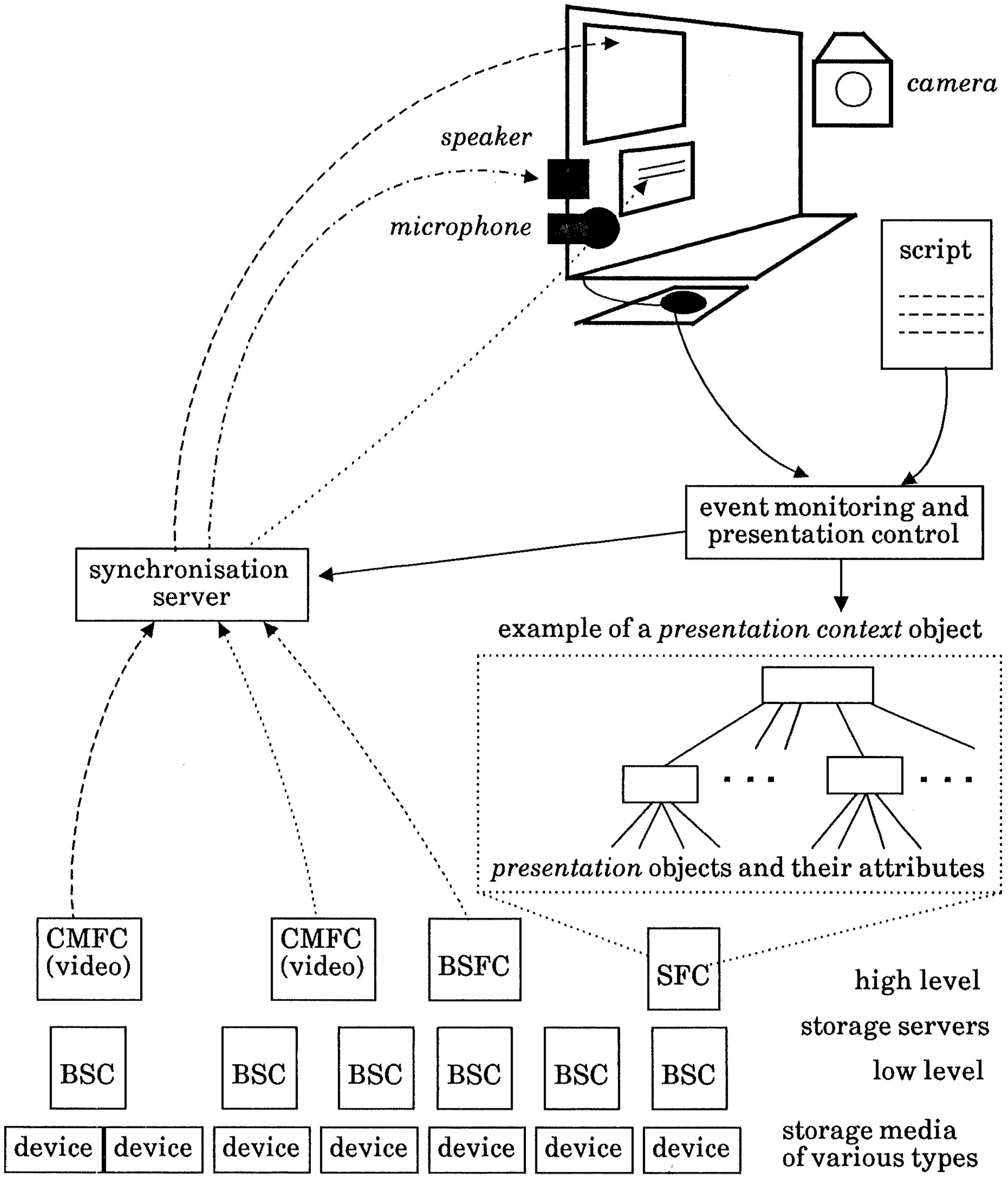


Figure 2 A multimedia presentation driven from storage by a script

4. Programming in PC++

We have developed a persistent programming language PC++, an extension to C++, for multimedia application programming [Wu 93], [Wu et al. 93]. A persistent class in PC++ keeps all the features of class in C, and provides additional features which are important for multimedia databases. It is integrated with the MSSA and uses the SFC (see Section 2.3) for storing both data and metadata, thus integrating the persistent programming language environment with the storage architecture.

PC++ supports the names (SSIDs) of objects managed within the open architecture of MSSA as a special type. Persistent class declarations can include SSIDs, and the application programmer may therefore construct and manipulate object data structures (for example, to represent a multimedia presentation) whose components are stored across many special purpose servers within the MSSA.

The use of the SFC for data storage has proved very convenient. Since SFC objects are tree-structured data migration and shadow versions may be managed at subobject level, which allows applications to work with **very large objects**. Also, PC++ stores the metadata table that maps its own persistent object identifiers (OIDs) to storage service identifiers (SSIDs) as a structured object on the SFC. This ensures that the **existence control** mechanism provided by the SFC applies even when there are multiple shadow versions of a (sub)object associated with current transactions.

Persistent objects can be located anywhere in the distributed system, and they are accessed as if they were local to the application. The use of remote procedure calls (RPCs) is hidden by a **remote object invocation** (ROI) mechanism provided by the system. The ROI mechanism also masks many of the failures that may occur in a distributed environment.

PC++ supports **transactions** (so that related changes may be made reliably to one or more objects) and uses an optimistic method of **concurrency control** (OCC). The concepts of *object* and *transaction* form an ideal basis for reasoning about the behaviour of distributed applications [Bacon and Moody 93], [Bacon 93, Part III]. An object model allows the semantics of an application to be used to specify the required concurrency behaviour of each object. A transaction model covers multi-component computations where the components are distributed and therefore subject to concurrent execution and partial failure. PC++ persistent objects have the properties of atomic data objects. A lower (physical) level guarantees the atomicity of method invocation. A higher (semantic) level enforces a serialisable execution that takes advantage of operation semantics.

OCC is appropriate for a multimedia environment in which concurrent write sharing is relatively rare. It is non-blocking; objects are not locked either during transactions or for lengthy atomic commitment procedures. OCC therefore does not interfere with guarantees of real-time access to objects. OCC also provides a model for continued working in the presence of server or communication failure and for planned detached working. The commit procedure for OCC used together with object semantics gives a coherent model for merging the results of a detached transaction with persistent system state.

5. Implementation

The storage service architecture MSSA is designed and implementations of the byte segment custode (BSC) and the structured file custode (SFC) are in everyday use. The presentation object store and active database mechanisms have been built above the SFC. The continuous medium file custode (CMFC) and the ticket mechanism are at the stage (March 93) of experimental testing.

A multimedia workstation, a Pandora box, is being used to test our ideas on representation of and support for multimedia presentations. Prototype versions of IMP and the active database are in use. A network-based synchronisation server has been built but has yet to be integrated with the storage and display systems.

To illustrate its functionality, PC++ has been used to reengineer a simple distributed application, namely to maintain the database for an active badge sytem that is used within the Laboratory. This is a low bandwidth application in which updates to the database are obtained from distributed collection points. The database can be interrogated from any terminal within the Laboratory.

6. Summary, further work and conclusions

The open storage architecture has given us flexibility and extensibility. Arbitrary file types at the high level are built on a common low level. Quality of service is supported in the form of guaranteed rates of acceptance and delivery of data which take compression into account. A two level architecture could be inefficient but this is avoided by the session and ticket mechanisms which allow the low level to be used directly but securely.

Our experience to date is that the object data model we use is supported well by the minimal structured data service, the SFC. We believe that the concept of an active database is appropriate for multimedia applications and we will evaluate this when we have more experience. We achieve a dynamic interactive display by applying an IMP script to a presentation context which is stored in a lightweight active database.

A persistent, object oriented programming language, with transaction support based on optimistic concurrency control, provides an ideal basis for programming reliable distributed multimedia applications. It also provides a good model for planned and unplanned detached working.

In summary, we have all the levels of a quality of service architecture to support multimedia applications designed and implemented, albeit in early prototype form. We will proceed to exploit, develop and evaluate its various components.

Acknowledgements

To Sue Thomson and Tim Wilson who wrote the first storage servers. To Heather Brown for many helpful discussions. To SERC for grant GR/H 13666 which supports the storage service aspects of the work. To Olivetti Research Laboratories, Cambridge, for network support and use of a Pandora box. To John Wilkes of HP Research Laboratories Palo Alto, CA, for support and continued interest.

References

[Bacon et al. 91]
Bacon J M, Moody K, Thomson S E and Wilson T D, "A Multi Service Storage Architecture" *ACM Operating Systems Review* 25(4): 47-65, October 1991

[Bacon 93]
Bacon J M "Concurrent Systems" Addison Wesley 1993

[Bacon and Moody 93]
Bacon J M and Moody K, "Objects and Transactions for Modelling Distributed Applications: Concurrency Control and Commitment" Computer Laboratory TR 293, 1993

[Bacon et al. 93]
Bacon J M, Bates J O, Lo S L, Moody K, " OPERA: Storage and Presentation Support for Multimedia Applications in a Distributed, ATM Network Environment" submitted to ACM SOSP14, 1993 and Computer Laboratory TR 295

[Bates 93]
Bates J O, "Support for Real-time Interactive Presentation of Distributed Multimedia" Computer Laboratory PhD thesis in preparation 1993

[Gong 89]
L. Gong, "A Secure Identity-Based Capability System", Proceedings of the IEEE Symposium on Security and Privacy, 56--63, May 1989

[Hopper 90]
Hopper A, "Pandora, an experimental system for multimedia applications" *ACM Operating Systems Review* 24(2): 19-34, April 1990

[Lo 93]
Lo S L, "Multi-service Storage Architecture" Computer Laboratory PhD thesis in preparation 1993

[Sreenan 92]
Sreenan CJ "Synchronisation Services for Digital Continuous Media" Computer Laboratory PhD thesis 1992

[Thomson 90]
Thomson S E, "A Storage Service for Structured Data" Computer Laboratory PhD thesis 1990

[Wu 93]
Wu Z, "A New Approach to Implementing Atomic Data Types" Computer Laboratory PhD thesis in preparation 1993

[Wu et al. 93]
Wu Z, Moody K M and Bacon J M "A Persistent Programming Language for Multimedia Databases" submitted to the Fourth Workshop on Database Programming Languages, New York, Aug-Sept 93,
and Computer Laboratory TR 296