INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND ASSOCIATED AUDIO

ISO/IEC JTC1/SC29/WG11
MPEG97/M1619
Seville, ES

Title:	**A Proposed Architecture for an Object-Based Audio-Visual Bitstream and Terminal**
Source:	**Alexandros Eleftheriadis and Hari Kalva** **Dept. of Electrical Eng., Columbia University, New York, NY 10027, USA** **{eleft,hari}@ee.columbia.edu**
Group:	**MPEG-4 System**
Date:	**February 2, 1997**

Abstract

This document describes an architecture for an object-based audio-visual terminal, expanding on the current design of the MPEG-4 System WD. The architecture assumes an object-oriented bitstream structure consisting of objects, composition information, and scene demarcation, which is described in detail in this document. The bitstream architecture allows on-line editing (cut and paste, insertions/deletions, grouping, special effects), and takes into account video-specific object structures (namely, predictive and interpolative object coding). This proposal eliminates the need to describe scenes by Java classes, thus decoupling the representation from any programmability requirements at the decoder.

Introduction

The current MPEG-4 architecture identifies two main components: a flexible and a non-flexible architecture. The non-flexible architecture is bitstream driven, in the sense that the decoder reacts to explicit “instructions” present in the bitstream. In the flexible approach, a scene is described by a Java class, which is responsible for retrieving bits form the bitstream. The former approach corresponds to a push-model, while the latter in a pull-model (in terms of which part is controlling information flow).

The flexible approach requires that the decoder is capable of executing Java VM code, and has several drawbacks:

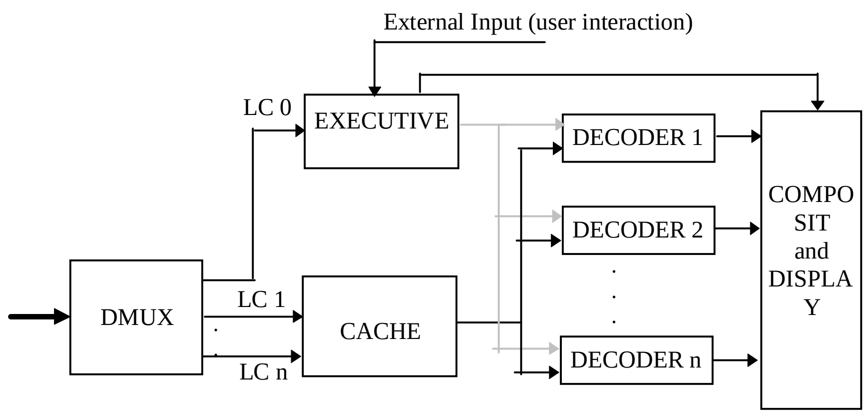
1. scene editing is difficult because the Java code has to be inverse engineered;
2. object insertion is very difficult, as it requires modification of the scene class, while it is impossible in real-time scenaria;
3. it is extremely difficult, if not impossible, to ensure adequate performance levels and guaranteeing proper operation of compliant implementations.

A fundamental limitation of the current architecture is that the scene description and the core engine which controls the overall operation of the terminal is essentially embedded in the downloaded scene description.

In the architecture presented here, the emphasis is placed in the bitstream itself. We believe that this is fundamental for an information representation standard such as MPEG-4. The details of the decoder's implementation are described in a a generic form, as was done in MPEG-2. Nevertheless, we should point out that the architecture described here is in no way incompatible with the current flexible architecture. What is described here is essentially object-oriented bitstream syntax, which can be easily accommodated in the current VM.

Overview

An audio visual (AV) terminal is an end user component used to present (display) audio visual content. An object-oriented terminal receives information in the form of individual objects, which are placed together to form a scene according to specified composition information. Objects and composition information are assumed to be transmitted in separate logical channels (LCs). The following figure shows the architecture of such a terminal.



In the AV terminal architecture presented, the AV objects and their composition information are transmitted (accessed) on separate logical channels. The DMUX receives the Mux2 layer from the lower layers and de-multiplexes it into logical channels. LC 0 always carries composition information which is passed on to the EXECUTIVE. The AV objects received on other logical channels are stored in the CACHE to be acted upon by the decoders. The EXECUTIVE receives the composition information, which includes the decoding and presentation time stamps, and instructs the decoders and compositor accordingly.

The system handles only two types of packets: object composition packets (OCP) and object data packets (ODP). A composition packet contains an object's ID, time stamps and the 'composition parameters' necessary to render the object. An object data packet contains an object ID, expiration time stamp in case of persistent objects and object data. The detailed structure of composition packet and data packet are shown in Fig 1.a and 1.b in the following pages.

Since the system handles only two types of packets, any external input (user interaction) should be converted to OCP and/or ODP before it is presented to the EXECUTIVE. This simple two packet approach also eliminates the need for any headers in a bitstream delivered over a network. However headers are necessary when storing a MPEG-4 presentation in a file.

Definitions

Object ID:

object number (10 bits)

XXXX.XXXX.XXXX.XXXX

object type (6 bits)

Object ID is composed of object type and object number. The default length of the Object ID is 2 bytes; ten bits for the object number and 6 for the object type (e.g. text, graphics, MPEG2 VOP, compound object). An extensible code is used to accommodate more than 1023 objects or more than 31 object types. The following convention shall be used in such cases. A value of 0b111111 in the first six bits of the ObjectID corresponds to: 31 + the value of the byte immediately following the ObjectID. A value of 0b11.1111.1111 in the least significant 10 bits of the ObjectID corresponds to: 1023 + the value of the two bytes immediately following the ObjectID (without counting the object type extension bytes, if present). The following object types are defined at this point:

0b00.0000	Scene Configuration Object
0b00.0001	Compound object
0b00.0010	Text
0b00.0011	MPEG2 VOP (rectangular VOP)
.	
.	
.	

Persistent Objects (PO): Objects that should be saved at the decoder for use at a later time. The life of a PO is given by an expiration time stamp (ETS). A PO is not available to the decoder after ETS runs out. ETS is given in milliseconds. When a PO is to be used at a later time in a scene, only the corresponding composition information needs to be sent to the AV terminal.

DTS: Decoding Time Stamp indicates the time an object (access unit) should be decoded by the decoder.

PTS: Presentation Time Stamp indicates the time an object (access unit) is presented should be presented by the decoder.

LTS: Lifetime Time Stamp, gives the duration (in milliseconds) an object should be displayed in a scene. LTS is implicit in some cases such as a video sequence where a frame is displayed for

1/frame-rate or until the next frame is available, whichever is larger. An explicit LTS is necessary when displaying graphics and text. An AV object should be decoded only once for use during its life time.

ETS: Expiration Time Stamp, is necessary to support the notion of object persistence. An object, after it is presented, is saved at the decoder (CACHE) until a time given by ETS. Such an object can be used multiple times before ETS runs out. A PO with an expired ETS is no longer available to the decoder.

OTB: Object Time Base defines the notion of time of a given AV object encoder. Different objects may belong to different time bases. The AV terminal adapts these time bases to the local one, as specified in the MSDL VM.

OCR: Object Clock Reference, can be used if necessary to convey the speed of the OTB to the decoder. This is the mechanism with which OTBs can be recovered/adapted at the AV terminal.

Composition Parameters: Parameters necessary to compose a scene (place an object in a scene). These include displacement from the upper left corner of the presentation frame, rotation angles, zooming factors, etc.

Priority: Indicated the priority of an object for transmission, decoding, and display. MPEG-4 supports 32 levels of priority. Lower numbers indicate higher priorities.

Persistence Indicator (PI): Indicates if an object is persistent.

Continuation Indicator (CI): Indicates the end of an object in the current packet (or continuation).

Object grouping: Object grouping is necessary in order to be able to apply operations to a set of objects with a single operation. This can be used to minimize the amount of composition information sent, as well as support hierarchical scene composition based on independent sub-scenes. Every compound object has a unique ID for handling at the decoder. The multiplex does not have an object with compound object ID but all the component objects are present in the multiplex. The compositor manipulates the component objects as a group.

Structure of a CCP:

OBJECT ID
DTS
PTS
LTS
OCR
COMPOSITION PARAMETERS

1.a Object Composition Packet

OBJECT ID
CI-PI
PRIORITY
ETS
VOP Type (I,P, B)
OBJECT DATA

1.b Object Data Packet

COMP. OBJECT ID
PTS
LTS
OBJECT COUNT
COMPOSITION PARAMETERS
OBJECT ID
DTS
OCR
COMPOSITION PARAMETERS

.
.
.

OBJECT ID
DTS
OCR
COMPOSITION PARAMETERS

1.c Compound Composition Packet

Object-Data packet:
ObjectID - min (default) 10 bits
CI and PI could be combined:

00	- Begin non-persistent
01	- Begin persistent
10	- continuation
11	- end of object

Priority: 5 bits, present only if CI/PI is 0b00 or 0b01
ETS: 30 bits, present if CI/PI is 0b01
For prediction based video coding, VOP_type is indicated by two bits (00 (I), 01 (P), 10 (B), 11 (PB)). This is to facilitate editing.

```
class ObjectID{
      uint (6)  object_type;
      uint (10) object_number;
}

class Object_data_packet{
   ObjectID objID;
   uint (2) CIPI
   if (CIPI <= 1){
      uint (5) priority;
      if (objID.type == prediction_based)
                     (any prediction based compression)
         uint (2) VOP_type;
   }
   if (CIPI == 1){
      bit (28) ETS;
   }
   ObjectData objData;
};
```

The class ObjectData contains transparent object-data the length of which is obtained from the mux layer. The precise class definition will be provided later.

Object Composition Packet:

```
class Object_composition_packet{
   ObjectID objID;
   bit (1) OCR_Flag;
   bit (1) Display_Timers_Flag;
   bit (30) DTS;
   if (OCR_Flag)
      bit (30) OCR;
   if (Display_Timers_Flag){
      bit (30) PTS;
      bit (30) LTS;
   }
   Composition_parameters compParam;
};
```

Composition Parameters:

The parameters in this structure are defined in section 2 of MSDL Verification Model [1].

```
class Composition_parameters{
    bit (1) visibility;
    if (visibility){
        uint (5) composition_order;
        uint (2) number_of_motion_sets;
        int (12) x_delta_0;
        int (12) y_delta_0;
        if (number_of_motion_sets > 0){
              int (12) x_delta_1;
              int (12) y_delta_1;
        }

        if (number_of_motion_sets > 1){
              int (12) x_delta_2;
              int (12) y_delta_2;
        }
        if (number_of_motion_sets > 2){
              int (12) x_delta_3;
              int (12) y_delta_3;
        }
    }
};
```

[1]. MPEG N1483: Systems Working Draft V2.0

Compound Composition Packet:

```
class Compound_composition_packet{
   ObjectID objID;
   bit (30) PTS;
   bit (30) LTS;
   Composition_parameters compParam;
   uint (8) ObjectCount;
   for (i = 0; i < ObjectCount; i++){
      Object_composition_packet objCompPckt;
   }
};
```

Scene Configuration Packet:

Scene Configuration Packet (SCP) is used to change reference scene width, height, to flush the cache, and other configuration functions. The object type for SCPs is 0b00.0000. This allows for 1024 different configuration packets. Only object number 0b00.0000.0000 is defined for this object type. The remaining object numbers are reserved and will be defined as necessary.

Use of Flush_Cache and Scene_Update flags: These flags are used to update the reference scene width and height whenever a new scene begins. If the Flush_Cache_Flag is set, the cache is flushed removing the objects if any from the cache. If Scene_Update_Flag is set, there are two possibilities: 1) Flush_Cache_Flag is set, implying that the objects in the cache will no longer be used. 2) Flush_Cache_Flag is not set, the new scene being introduced (an editing action on the bitstream) splices the current scene and the objects in the scene will be used after the end of the new scene. The ETS of the objects, if any, will be frozen for the duration of the new scene introduced. The beginning of the next scene is indicated by another scene configuration packet.

```
class Scene_configuration_packet{
    ObjectID objID;
    objID.object_type = 0;
    objID.object_number = 0;

    bit(1) Flush_Cache_Flag;
    bit(1) Scene_Update_Flag;
    if (Scene_Update_Flag){
          uint(12) ref_scene_width;
          uint(12) ref_scene_height;
    }
};
```

References

[1] ISO/IEC JTC1/SC29/WG11 N1483, MPEG-4 Systems Working Draft Version 2.0, Maceiò, Brasil, Nov. 1996.

[2] ISO/IEC JTC1/SC29/WG11 N1484, MPEG-4 Systems Verification Model Version 2.0, Maceiò, Brasil, Nov. 1996.

[3] ISO/IEC JTC1/SC29/WG11 N1378, MPEG-4 Audio Verification Model 2.0, Chicago, USA, Sept.-Oct. 1996.

[4] ISO/IEC JTC1/SC29/WG11 N1469, MPEG-4 Video Verification Model 5.0, Maceiò, Brasil, Nov. 1996.

[5] ISO/IEC JTC1/SC29/WG11 N1454, MPEG-4 SNHC Verification Model 5.0, Maceiò, Brasil, Nov. 1996.