ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May 28-31, 2000, Geneva, Switzerland

Interactive Broadcast Digital Television

The OpenTV Platform versus the MPEG-4 Standard Framework

Frederic Bouilhaguet, Jean-Claude Dufourd, Souhila Boughoufalah,
*Christophe Havet **
ENST Paris, dept ComElec - 46 rue Barrault – 75013 Paris, France
** OpenTV Europe – 160 bis, rue de Paris – 92645 Boulogne Cedex, France*
E-mail: {bouilhaguet, dufourd, souhila}@enst.fr, chavet@opentv.com

Abstract

The last years have seen a vast increase in Interactive Broadcast Digital TV systems featuring proprietary platforms coupled with the standard MPEG-2. OpenTV is an example of a proprietary platform present on the market. The study of the OpenTV multimedia delivery framework and our involvement in MPEG-4 normalisation works give us the occasion to compare the OpenTV system architecture with the MPEG-4 one to address 2D interactivity in Broadcast Digital TV applications. We study their respective approaches to define 2D user interfaces and to manage dynamic interactive audio-visual contents.

1. Introduction

Changes in the nature, scope and extent of multimedia on-line applications are very rapid, due to the major developments in computing and telecommunications technology. Broadcast Television has undergone important evolutions since it started to be digital. After broadcast digital channels with only audio and video (MPEG-2), the growing tendency is to develop technologies for interactive contents based on push/pull scenarios. Starting from the only video and audio model standardised by MPEG-2, the Broadcast Digital TV industry has provided new systems using objects description techniques to afford the bases for developments of interactive services. The transport structure of MPEG-2, called Transport Stream (TS), allows to multiplex with the MPEG audio and video signals any other type of digital stream, the so-called "private data". So the market didn't wait for a new version of MPEG standard to provide new system architectures for the development of interactive programs compatible with MPEG-2 TS. The definitions of these system architectures consisted in specifying structures of new streams, as shown in the *figure 1,* and their corresponding codecs to transport the spatio-temporal descriptions needed for interactivity.
The OpenTV platform provides one of these system architectures. It is based on a new *opentv* stream added to *MPEG2_audio* and *MPEG2_video* ones. The *opentv* stream transmits OpenTV applications that are computer programs. The OpenTV applications are currently developed in ANSI C and compiled with a special development kit compiler. The output from the compiler is called *o-code* and consists of a private byte code that is interpreted by the *o-code* interpreter and executed on the digital interactive decoder. OpenTV applications make *o-code* function calls to OpenTV library. The library routines initiate operations or requests. The implemented Application Programming Interface (API) of this library determines the structure and rules of composition of the visual and audio objects.

Today, the Quality of Service (QoS) is starting to suffer from serious problems of incompatibilities between all the multiple system architectures provided by different companies. Final users must have several decoders to decode interactive channels provided by different broadcasters who chose different system architectures. Content producers must redevelop the same interactive audio-visual application to suit all the platforms.
MPEG with the new MPEG-4 standard version aims at specifying an open system architecture adaptable to potential future platforms of Broadcast Digital TV and suppressing previous sources of incompatibility. The first component of the MPEG-4 system architecture to fit the requirements is two new elementary streams: the Binary description Scene Format (BiFS) stream and the Object Descriptors stream. An MPEG-4 scene is a set of organised audio and visual objects with behaviours that constitutes a part of a 2D interactive application. BiFS is the compressed format in which scenes are defined and modified. BiFS is composed of 2D and 3D profiles. Our interest here is 2D only.

In this paper, we show how OpenTV and MPEG-4 differ in their common requirements for:

- Composing interactive user interfaces from object description techniques (section 2)
- Communicating events between objects (section 3)
- Having access to multiple audio and video data sources (section 4)

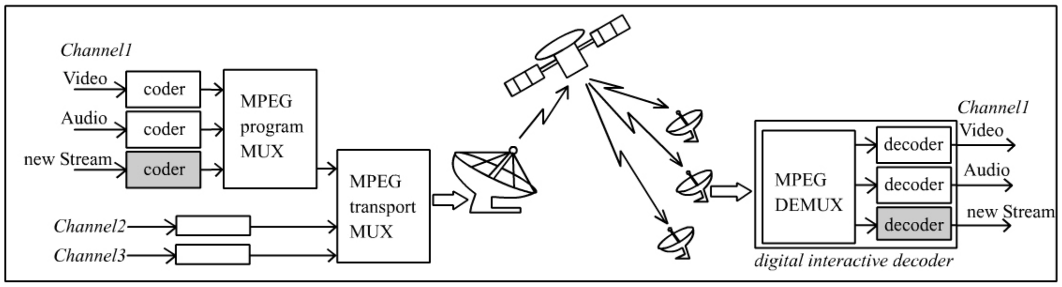

Figure 1: typical transmission chain of broadcast interactive channels

2. Gadget tree versus scene graph

OpenTV and MPEG-4 describe objects and their behaviour in hierarchical models. MPEG-4 uses the concept of a *scene graph* with *object nodes*, while OpenTV prefers a *gadget tree* with *gadgets*.

2.1 Scene graph (MPEG-4)

The graph and its drawing order:
An MPEG-4 scene is constructed as a direct a-cyclic graph of nodes.
The following types of nodes exist:
- Grouping nodes construct the scene structure;
- Children nodes are the children of grouping nodes that represent the multimedia objects in the scene.
- Interpolator nodes are another subtype of children nodes that represent interpolation data to perform key frame animation. These nodes generate a sequence of values as a function of time or other input parameters.
- Sensor nodes sense the user and environment changes for authoring interactive scenes.

BiFS scenes are composed of a collection of nodes arranged in a hierarchical tree. Each node represents, groups, or transforms an object in the scene and consists of a list of fields that define the particular behaviour of the node.
The root node of a 2D BIFS scene can be an OrderedGroup or a Layer2D node. OrderedGroup may be used to specify the drawing order of elements of the scene because this grouping node controls the visual layering order of its children.

Reusing objects:
BIFS has a mechanism for reusing nodes. For example, once a complex graphic object is defined as a collection of geometric primitive nodes collected inside a Group node, it is possible to reuse the object elsewhere in the scene, rather than copying it explicitly wherever it is to appear. Each reusable node has a binary ID, and wherever a node can appear in the scene description, the ID of a reusable node can be inserted.

The prototypes:
BIFS provides ways to encode PROTO and EXTERNPROTO. These scene constructs enable the definition of new interfaces to user-constructed scene components. For example, a button PROTO can be constructed which accepts a string label as an input parameter. The body of the PROTO consists of a scene portion that draws a button (using, say, a Box node) and renders the string parameter on the button. The EXTERNPROTO is similar to the PROTO, except that its definition is not part of the scene. Instead, its definition is referenced using a URL. This enables the construction and use of on-line libraries of PROTOs.

2.2 Gadget Tree (OTV)

The tree and its drawing order:
OpenTV provides an object-oriented framework for defining classes of user interface elements called *gadgets*. A *gadget class* specifies the behaviour functions for all gadgets of the same class. Gadgets are created and combined by an OpenTV application to form its user interface. They are drawn on the screen in a predictable order of a tree structure, as shown in the *figure 2*. The position of a gadget in the tree determines the order in which it is drawn. Drawing starts at the root, and descends the left-most branch first, then traverses that branch, then moves onto the sibling just to the right.
At any given time, the interface manager of the OpenTV operating system recognises one gadget to be the root of the sub-tree currently being displayed on the TV screen. This is set by calling the *O_ui_set_root* function. The root may change over the lifetime of the application if it is necessary to display a new screen. Gadgets must belong to the sub-tree owned by the designated root to be visible on the TV screen. There are functions for creating, activating (making visible), deactivating (making invisible), deleting gadgets. The *O_gadget_attach* function attaches a new child gadget to a specified gadget.

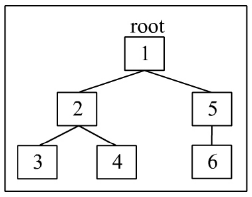

Figure 2: OTV gadget tree drawing order

```
/* Create some button gadgets and attach them to 'this' group. */
while (nextItem != null) {
 o_gadget next_Button =   O_gadget_new_from_resource(*nextItem);
 nextItem++ ;
 O_gadget_attach(this,next_Button);
}
```

Reusing gadgets:
The *O_gadget_new* function creates new *gadgets*, i.e. instances of a *gadget class*. A *gadget* can be reused in its application scope until it is removed with the *O_gadget_delete* function.

The prototypes:
There is nothing here equivalent to what MPEG-4 calls prototypes. Each gadget class is in fact prototyped. Gadget class's developers define a structure for passing initial value and support functions to supplement the functionality of the gadget class.

3. Events communication

3.1 Sensors and routes (MPEG-4)

The node fields are labelled as being of type *field*, *eventIn*, *eventOut*, or *exposedField*. The *field* label is used for values that are set only when instantiating the node. Fields that can receive incoming events have the *eventIn* label, whereas fields that emit events are labelled as *eventOut*. Finally, some fields may set values but also receive or emit events, in which case they are labelled as *exposedField*
To describe interactivity and behaviour of scene objects, the MPEG-4 event architecture defined sensors and routes. Sensor nodes generate events based on user interaction on a trigger node or a change in the scene. An event can be routed from any sensor node *eventOut* field to interpolator or other nodes to change the attributes of these target nodes. If routed to an interpolator, a new parameter is interpolated according to the input value, and is finally routed to the target node eventIn field. This target node processes the event. The scope of a sensor is delimited to the children of the grouping node that contains it. In other words, routes are used to propagate events between scene elements. They are connections that assign the value of one field to another field.
The following figure 5 and the BIFS text of 4.2 shows an example of how to trigger an action as the cursor rolls over a button object.

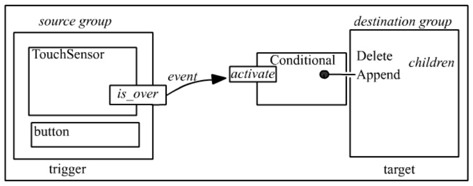

Figure 5: example of transmission of events in BIFS

3.2 Messages handlers (OpenTV)

To support input processing, OpenTV have the notion of *focus*. Only one gadget in the tree is designated as having the *focus*. All input will be directed to this gadget. The gadget is notified of user input by receipt of messages of the appropriate types. If a gadget ignores an input message, it will propagate up to that gadget's parent, and so on, so processing of input is also affected by a gadget placement in the gadget tree.

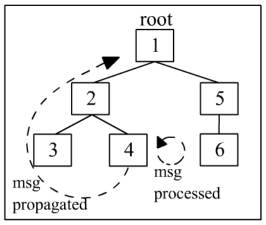

Figure 3: transmission of messages in the gadget tree

At the initialisation of a gadget class, the function that handles events called *message_handler* is passed as an argument. This *message_handler* function is called each time a message (MSG_TYPE_NEW / DELETE / ACTIVATE / DEACTIVATE) is send to a gadget of this class. A key-pressed event is sent only to the gadget that has the focus. If the gadget doesn't use the key event, the event is automatically routed to its parents recursively until its associated message is consumed by calling the *O_ui_msg_used function*.

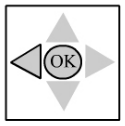

Figure 4: remote control

In the following example, navigation between buttons is done by pressing the left arrow key and an action is performed when the enter *"OK"* key is released on the button, as seen in the figure 4. The ancestors of the button gadgets may handle other keys.

```
/* The events handler function of the button gadget */
static void button_message_handler(button* this,o_message* msg)
{
  switch (O_msg_type(msg))
  {
   ......
   /* A key has been pressed */
   case MSG_TYPE_KEY_DOWN:
     button_key_down(this,msg);
     break;
   ......
  }
}

/* Handle key down events */
static void button_key_down(button* this, o_message* msg)
{
  ......
  switch(O_ui_msg_key(msg))
  {
     /* Handle the selection key */
     case KEY_ENTER_CODE:
     /* Draw something to show
        that the button is pressed  */
      ......
      /* Letknow we consumed the key event */
      /* The message is not passed to its father */
       O_ui_msg_used(msg);
       break;

     /* Pass the focus to its left button
       (in the gadget tree) */
     case KEY_LEFT_ARROW_CODE:
     {
       next_button = O_gadget_left_brother(this);
       /* Only change focus if a left button exists */
       if (next_button!=NULL)
         O_gadget_set_focus(next_button);
       /* Let know that we consumed the key event */
       /* The message is NOT passed to its father */
       O_ui_msg_used(msg);
       break;
     }
     ......
  }
    }
```

4. Access to Audio/Video streams

4.1 Object Descriptor (MPEG-4)

MPEG-4 defines an ObjectDescriptor (OD) stream. It is coupled with the BIFS stream to identify and describe elementary streams that are associated to the audio-visual scene description. An OD is a collection of one or more elementary stream (ES) descriptors. It is assigned an identifier (object descriptor ID). An ES descriptor include information about the source of the stream data and encoding format for the decoding process.
Here is an example of how we can switch with BIFS from a current audio source to a new one when the cursor rolls over a button. The BIFS audio node points to the audio data stream through the OD's ID value contained in the url field. The audio is switched by removing the current audio stream from the grouping node and appending the new one.

BIFS text:

```
DEF ID_100 Group{
 children [
  # **** original sound
  Sound2D {
   source audiosource {
     url 5 # *** object descriptor ID
     startTime 0
     stopTime -1
   }}
DEF ID_211 Conditional {
 buffer {
   # **** remove the original sound
   DELETE ID_100.children[0]
   # **** switch to the new sound
   APPEND TO ID_100.children
    Sound2D {
     source audiosource {
      url 6 # *** object descriptor ID
      startTime 0
      stopTime -1
     }}}}
# **** hot spot area :
Group {
 children [
  DEF I205_ TouchSensor {}
  Transform2D {
   translation pos_X pos_Y
   children [
     Shape { geometry Rectangle { size w h }
             appearance Appearance {...}
     }]}]}
# **** propagate events
ROUTE I205_.isOver TO ID_211.activate
...
```

ObjectDescriptor of the audio stream :

```
{
  objectDescriptorID    6
  esdescr [
    ES_Descriptor {
      es_id 3
      decConfigDescr DecoderConfigDescriptor {
        objectTypeIndication    0x40
        streamType      5
        upStream        FALSE
        bufferSizeDB        8000
        maxBitrate      0
        avgBitrate      0
      }
      slConfigDescr SlConfigDescriptor {
      ... #long sync layer parameter list
      }}]}
```

4.2 Station control (OTV)

OpenTV programs can switch from elementary streams via the *O_station_control* function. A call to this asynchronous function enables to open/close elementary streams. The parameter of *O_station_control* is an actions list so several actions can be performed in a single call. When an action is completed, a message is posted in the events queue. The different streams supported are video (*O_VIDEO*), audio (*O_AUDIO*), opentv (*O_OPENTV*), subtitle (*O_SUBTITLE*), and teletext (*O_TELETEXT*).
Here is an example of switching audio tracks by pressing a button gadget.

Switch to a new audio stream in the C file:

```
/* actions list */
char switch_to_audio_1[20] = {
  C_STREAM_ON, 5,
  O_AUDIO, '0', '0', '1', 0,
  C_END, 0 };
......
/* Application main loop */
for (;;){
 /* Get message from the events queue */
 O_ui_get_event_wait (&msg);
 switch( O_msg_class(&msg) ) {
  ......
  case audio1_button_ID:
    O_station_control(switch_to_audio_1);
    break;
  ......
}}
```

Description of the audio stream in the transport stream configuration file:

```
elementary_stream   {
  stream_type = audio_mpeg2
  elementary_stream_pid = 523
  descriptor {
    descriptor_tag = iso_639_language
    language {
      language = "001"
      audio_type = 0}}}
```

5. Other aspects

5.1 MPEG-J

The BIFS stream offers a parametric scene representation while OpenTV rather offers a programmatic environment. MPEG-4 standard will also offer a programmatic environment, in addition to its parametric capability. Java APIs are defined and a dedicated *MPEG-J* stream can be added to BIFS and OD streams. Access to an underlying MPEG-4 engine can be provided to Java applets, called MPEG-lets. MPEG-J forms the basis for very sophisticated applications, opening up completely new ways for audio-visual content creators to augment the use of their content. MPEG-J will be available in MPEG-4 version 2.

5.2 Transparency and Composition

The composition approach in OpenTV differs from MPEG-4. From the use of functions such as *O_palette_set* and *O_palette_set_transparency*, the current color lookup table (palette) is set and transparency can be applied to some of the indexed colors. MPEG-4 has a *transparency* field in the *Appearance* node that can be associated with any visual object, even video or still pictures. So transparency is applied per object and color space is not limited to a palette.

6. Conclusion

We have presented briefly two systems for Interactive Broadcast Digital TV, OpenTV and MPEG-4. The OpenTV environment is far from the MPEG-4 delivery framework if we just consider BIFS and OD streams, even if we saw that some common approaches existed in the two ways of organising visual objects. If we consider MPEG-J, MPEG-4 moves toward the OpenTV programmatic approach. One perspective for the OpenTV framework could be to implement on top of the OpenTV engine the MPEG-J APIs with a Java VM in order to provide a first MPEG-4 terminal compliant with MPEG-J so that future MPEG-lets can execute on the OpenTV platform.
There are pros and cons for both: on paper, MPEG-4 is much more powerful but OpenTV is here now.

References :

[1] "MPEG-4 Systems FDIS", Document ISO/IEC JTC1/SC29/WG11/N2501, Atlantic City MPEG meeting, October 1998

[2] "MPEG-4 Visual FDIS", Document ISO/IEC JTC1/SC29/WG11/N2502, Atlantic City MPEG meeting, October 1998

[3] "MPEG-4 Audio FDIS", Document ISO/IEC JTC1/SC29/WG11/N2503, Atlantic City MPEG meeting, October 1998

[4] "MPEG-4 DMIF FDIS", Document ISO/IEC JTC1/SC29/WG11/N2506, Atlantic City MPEG meeting, October 1998

[5] MPEG Requirements Group, "Overview of MPEG-4 Profile and Level Definitions", Document ISO/IEC/JTC1/SC29/WG11/N2458, Atlantic City MPEG meeting, October 1998

[6] MPEG Requirements Group, "MPEG-4 Version 2 Overview", Document ISO/IEC/JTC1/SC29/WG11/N2324, Dublin MPEG meeting, July 1998

[7] OPEN TV, "Software Developer's Kit 2.0 - SoftwareDeveloper's, Mountain View, March 1999

[8] OPEN TV, "Software Developer's Kit 2.0 – Software Developer's Reference Manual, Mountain View, March 1999