Hindawi Publishing Corporation
International Journal of Digital Multimedia Broadcasting
Volume 2008, Article ID 319063, 18 pages
doi:10.1155/2008/319063

Research Article

Real-Time DVB-MHP Interactive Data Transcoding to Blu-Ray

Sergio Infante and Panos Nasiopoulos

Department of Electrical and Computer Engineering, The University of British Columbia, 2332 Main Mall, Vancouver, BC, Canada V6T 1Z4

Correspondence should be addressed to Sergio Infante, sergioi@ece.ubc.ca

Received 15 April 2008; Revised 30 June 2008; Accepted 23 August 2008

Recommended by M. Roccetti

Digital TV systems are being deployed worldwide, and interactive applications are being offered as part of the services. The unparalleled success of DVD technology has motivated the development of interactive services for broadcast TV, with DVB-MHP being the most widely used open standard in the world. Despite the similarities between DVB-MHP-based interactive TV and the new Blu-ray format, there still exist substantial differences that make the two systems incompatible. In this paper, we analyze the differences in the DVB-MHP and Blu-ray interactive formats and propose a transcoding scheme to convert live broadcast interactive TV to a Blu-ray compatible format.

1. INTRODUCTION

Digital television systems offer many advantages over analog systems, such as higher definition and interactivity and are expected to replace the 50-year-old analog TV systems in the next few years. This change has brought about a reconceptualization of services and choices. Digital media systems allow video, audio, and data to be integrated in a single package or TV service, so that media producers and distributors can offer an enhanced experience to the consumer. A very successful example of this integration is the DVD-Video, which provides multiple enhanced capabilities through additional streams that complement the primary video and audio streams and data that support navigation and menu-based interactivity. The success of DVD-Video motivated the development of interactive services for broadcast TV. Current broadcast interactive TV (iTV) provides different types of interactivity by enhancing the TV signal with applications, consisting of software and complementary data. Current iTV set-top boxes are enabled to receive and execute the software applications that are broadcast along with the audio and video.

Several iTV standards have emerged over the past decade. In 2000, the digital video broadcast (DVB) group released the multimedia home platform (MHP) [1, 2]. This standard, better known as the DVB-MHP open standard, specifies the format and syntax for broadcasting data and software and defines a set of software libraries that must be included in the execution engine of the TV set. The proposed execution engine uses Java technology in an effort to support a platform-independent execution environment [3, 4]. iTV systems based on this standard have been deployed around the world, and DVB-MHP is currently the most widely used open standard [5]. The DVB group then developed the globally executable MHP (GEM) specification [6] based on the DVB-MHP specification. GEM has been used as the basis for the development of other iTV specifications such as the open cable applications platform (OCAP) [7] and the advanced common application platform (ACAP) [8].

More recently, DVD technology has also taken a step forward. One of the most recent DVD specifications for supporting high-definition media, the Blu-ray system, has adopted the GEM specification as the foundation for its interactive applications [9]. This approach allows for interactive features far beyond those offered by the traditional DVD, and a positive move toward convergence of broadcast iTV and prerecorded media. However, despite the similarities between the DVB-MHP and Blu-ray standards, substantial differences remain that make the two systems incompatible. On one hand, in a broadcast environment, interactive data

(iTV software applications) are transmitted along with audio and video content in a single stream. In this case, interactive data may be encoded and encapsulated according to the DVB-MHP broadcasting standards. On the other hand, in the Blu-ray system, interactive data are stored separately from the audiovisual content. The audiovisual content is stored in a format different from the broadcasting transport stream format, and the interactive data are stored in a single archive, which is different from the encapsulation format used in broadcast iTV.

The increasing number of choices in digital services available to the consumer brings with it an increased number of required access devices. For iTV, a set-top box is needed in order to tune, decode, and play iTV. This box and its remote controller will be added to the current set of TV-related devices in many homes, which typically consists of a TV display, tuner device, audio system, and a DVD recorder/player (soon to be replaced by a Blu-ray player). Adding an iTV player to this set of devices not only requires physical space and adds complexity to an already unmanageable entertainment system, but it also forces the end user to learn a new set of functions in order to operate the new device.

The advances in DVD technology and broadcast iTV and their convergence to common interactivity paradigms point to a possible simplification of this setup. It is, therefore, highly desirable to eliminate the added complexity by converting the real-time iTV content to a Blu-ray compliant format, so that an additional set-top box is not required. Such an effort would include transcoding of the interactive data, the audiovisual data, and the system information from DVB-MHP to Blu-ray. Up to this date, and to the best of our knowledge, no work has been published on converting iTV interactive data to the Blu-ray format.

In this work, we focus on the development of a transcoding scheme that efficiently converts DVB-MHP interactive data to Blu-ray compliant format. This is the first in-depth analysis of the differences between the DVB to Blu-ray standards, which leads to the first attempt to convert the real-time iTV data to Blu-ray format. In our study, we do not propose conversion schemes from one standard to the other, but we rather offer transcoding solutions that aim at reducing the complexity and computations of the conversion process, resulting in a real-time implementation.

The rest of this paper is organized as follows. Section 2 presents background information related to iTV and Blu-ray and also reviews work related to iTV signal processing. Section 3 compares the two different standards in relation to the interactive data of each and presents our proposed transcoding scheme. A performance evaluation of our method and discussion of results are presented in Section 4. Section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK

The multimedia home platform (MHP) is the DVB group's specification for iTV. In this specification, an iTV program consists of Java applications transported together with audio and video contents [10]. The DVB-MHP establishes recommendations for iTV broadcasting as well as for applications programming. The iTV broadcasting recommendations are based on the standards previously defined by DVB [11]. For the applications programming, the DVB-MHP standard borrows the application model and lifecycle from the JavaTV technology [1].

The demand for interactive services pushed the DVB group to develop a new standard for iTV: the globally executable MHP (DVB-GEM). Although this specification is based on the DVB-MHP standard, it does not define any broadcasting mechanisms. For this reason, it can be implemented in networks that do not follow the DVB broadcasting standards. In DVB-GEM, two main groups of iTV are identified based on the way the content is transported from the producer to the viewer: (1) prerecorded media (which is called "packaged media" in the specification), where the content is prerecorded onto a physical carrier, such as a disc or tape and (2) broadcast media, which refers to broadcast environments like cable or terrestrial TV [3].

2.1. Broadcast interactive television

iTV content is delivered to the receiver end in the form of a transport stream. This stream contains audio and video bitstreams that are encoded according to the formats specified in the MPEG-2 standards [12, 13]. It also contains applications which in their basic form are sets of files organized in directory structures. The basic format and syntax of the transport stream are established in the ISO/IEC 13818-1 specification (MPEG-2 systems) [14]. The content and basic format of the stream are depicted in Figure 1.

The audio and video bitstreams are split into packetized elementary stream (PES) packets. Content other than audio and video, such as the interactive data, is split into special packets known as sections. Each PES packet and each section are split into smaller packets of 188-byte length, which in turn are inserted in the transport stream (TS). Thus, the transport stream is a sequence of TS packets. Each TS packet consists of a 4-byte header and a payload (a fragment of a PES packet or section). The header contains a packet id (PID) used for packet content identification (see Figure 1).

The MPEG-2 TS can carry multiple programs. A program is a set of associated video, audio, and data with a common time base [14]. A program is also known as *service* in the DVB specifications [15–17]. The multiple programs in TS are supported by a set of tables, known as service information (SI), which are broadcast in the stream and contain data about the stream's content. The MPEG-2 standard defines 4 such tables: the program association table (PAT), program mapping table (PMT), conditional access table (CAT), and network information table (NIT). In addition to the service information tables (SIT) established in MPEG-2, a table that provides the receiver with information regarding the interactive applications is included in the iTV stream. This table is known as application information table (AIT) and it contains a description of the applications in the stream, as well as management information.

All the SI tables are carried in MPEG-2 sections and follow the standard MPEG-2 service information table (SIT)

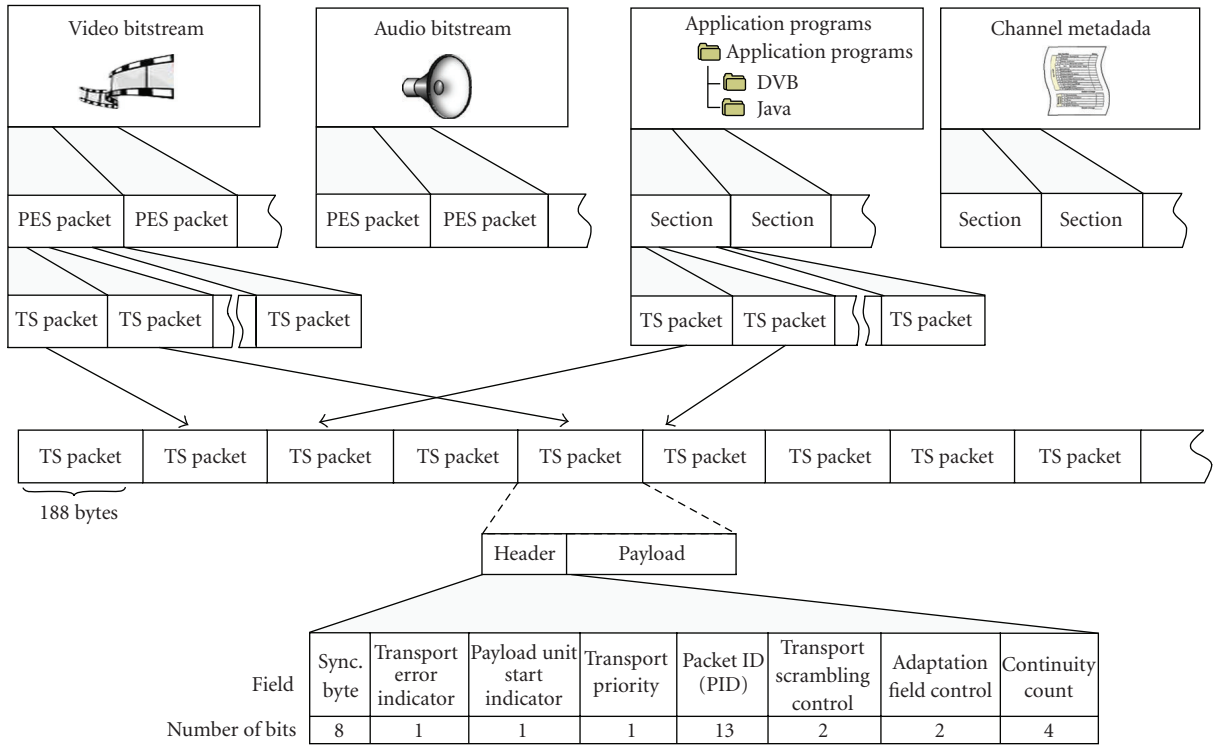

Figure 1: Detailed structure of the MPEG-2 transport stream (TS).

format. According to this specification, the table consists of *fields* and *descriptors*. A *descriptor* groups *fields* into sets that have a common task. For instance, when the *fields* provide information related to language, the fields are grouped together, and the associated *descriptor* is the name of the language (e.g., English). Some of the *fields* may not be associated with a *descriptor*. These fields are known as *basic fields*. Different descriptors are established in the standards and specifications documents [1, 6, 14, 17]. Some of them are optional. In every case, basic fields are mandatory.

The interactive data in an iTV program are structured as a set of files which constitute the interactive application (e.g., an executable software program). Some of these files contain executable code known as byte code. This byte code is executed by the Java Virtual Machine that is part of the system software of the iTV receiver. Other files contain information that is used as a supporting resource, for instance, background images and text information for news updates. All these files are organized in directories whose structure is similar to the one shown in Figure 2.

In DVB-MHP, the directory containing the files of an application is formatted according to the broadcast interorb protocol (BIOP), also known as the object carousel protocol [18–25]. Every file and directory is encapsulated in a BIOP message, which consists of a message header and a subheader (see Figure 3). The fields in the BIOP message header describe general message parameters, such as the size of the message, while the BIOP subheader fields describe parameters specific to the type of object in the message. For example, in the case of directory objects, the corresponding

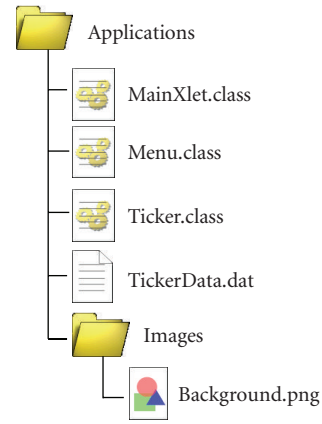

Figure 2: Example of the organization of interactive application data.

fields indicate the number of elements in the directory. BIOP messages are grouped in modules with a maximum size of 64 KB. Each of these modules splits into smaller blocks known as data blocks. These data blocks are encapsulated in DSM-CC *download protocol* messages that are in turn encapsulated into sections in order to transmit the object carousel data. The transmission of such object carousel data is performed cyclically.

In the DSM-CC object carousel, there are two categories of download protocol messages: the control messages and the data messages. The control messages are used by the

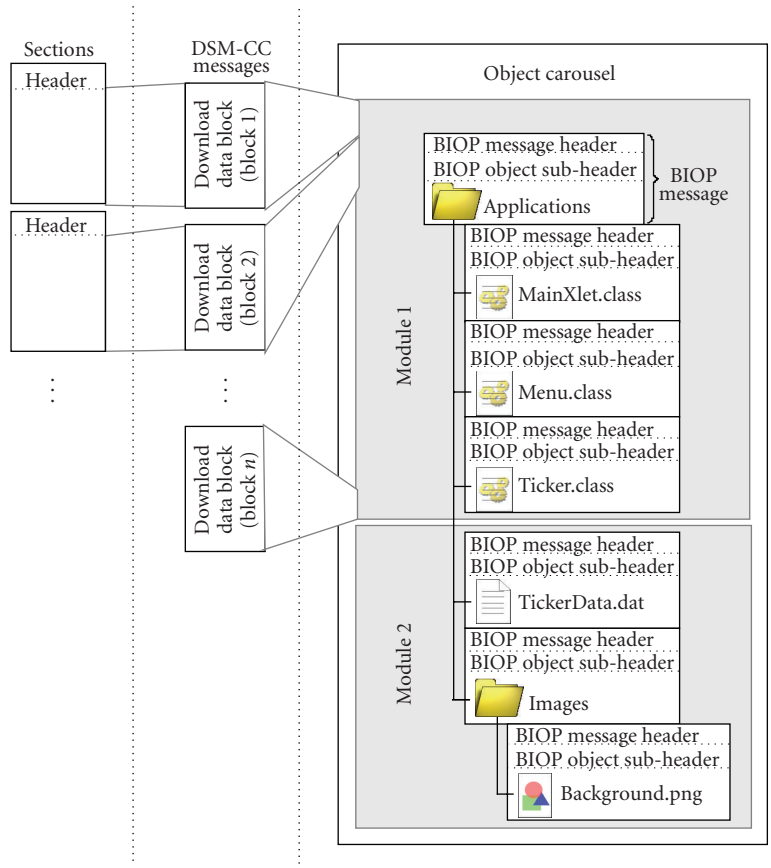

Figure 3: Interactive data format in DVB-MHP.

Table 1: DSM-CC download protocol messages.

Message	Function	Category
Download server initiate (DSI)	Advertise or identify a server (broadcaster)	Control message
Download info indication (DII)	Inform the client download parameters about the modules to be transmitted	
Download cancel	Prematurely terminate the transmission	
Download data block (DDB)	Transfer data from a server to client	Data message

broadcasters to provide information regarding transmission (i.e., number of modules to be transmitted and size of each module), while the data messages are used to transfer the actual data of the modules. Table 1 lists the different messages for the two categories found in an object carousel, as well as their corresponding functions.

In order to specify resources that broadcast iTV programs can access (e.g., a video stream), the DVB group defines a special type of string known as *locator*. A *locator* is a string that uniquely identifies a resource. In DVB-MHP, the basic *locator* string includes DVB specific identifiers such as the *network id* (provider of the TV services), the *transport stream id* (broadcast channel in the TV network), and the *service id* (a specific program in the transport stream), as shown below

$$\text{dvb://network\_id.transport\_stream\_id.service\_id.} \quad (1)$$

The *network id*, *transport stream id,* and *service id* are pointers to the actual address which resides in the service information tables. An example of an actual DVB locator is

$$\text{dvb://122.164.73,} \quad (2)$$

where 122, 164, and 73 are the *network id*, *transport stream id,* and *service id* in hexadecimal notation. This basic locator identifies a specific service (a program in a broadcast channel), which contains a unique name for a resource. It is based on data that can be read from the service information tables.

2.2. Blu-ray

The Blu-ray format has been developed by the Blu-ray disc association. According to the Blu-ray specification, the Blu-ray playback system is able to work in two different modes. The first mode, known as HDMV, deals with high-definition

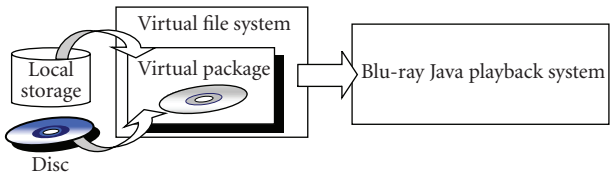

Figure 4: Simplified view of the Blu-ray playback system input.

movie playback and provides functions similar to those provided by the DVD-video, with some improvements in text subtitle display, multipages, and pop-up menus. The second mode in the Blu-ray playback system is oriented to interactive applications and is known as Java mode or Blu-ray Java. This mode is based on the globally executable MHP (GEM) [6, 9]. Because of this, the Blu-ray Java playback system has most of the elements needed to execute DVB-MHP applications. The architecture of the Blu-ray Java playback system is based on the basic architecture of the DVB-MHP playback system.

In the Blu-ray system model, there are several input elements, such as the Blu-ray disc and local storage. All different input elements are abstracted by a layer called virtual file system (VFS). This layer presents a simplified and unified view of the input elements to the playback system. The content in the local storage, optical disc, and any other storage devices is combined into a virtual package (Figure 4), in such a way that the application environment is not aware if the data are stored on local storage, the optical disc, or other input device.

A Blu-ray program (known in DVD terms as *title*) consists of video, associated audio, metadata (data that describes the content), and Java applications. In contrast to DVB-MHP, where the program components are stored in the same stream in Blu-ray, the program components are stored in different files. These separate files of the same program are linked by metadata and database files. Every program is registered in an index table. Each program in the table is linked to an object that contains management information associated to the program. The object is linked to audio and video files, as well as to the files that contain the interactive application data (Figure 5).

The audio and video files are formatted according to the Blu-ray audio video (BDAV) stream structure. The BDAV stream, depicted in Figure 6, is based on the MPEG-2 transport stream. In a BDAV stream, the MPEG-2 TS packet is extended by adding an extra header, and the packets are organized in groups called aligned units. The extra header contains a time stamp, which allows content indexation in Blu-ray (i.e., scene selection).

A BDAV stream may contain several streams of audio, video, text, and graphics. For this reason, the basic MPEG-2 service information tables (SITs) are also included in the stream to provide information about its content. However, the content of these tables is restricted. For instance, the values of the PIDs for the different types of content are restricted to a range specified in the Blu-ray standard. The functional equivalent to the DVB-MHP's AIT is the application management table (AMT) [9]. This table describes one or more applications in a program. Although both the AIT DVB table and the AMT Blu-ray table are used for the same purpose, they differ in the way data is organized inside each table.

The interactivity data for a Blu-ray Java title is stored in what is known as jar file. The jar files are the functional equivalents to object carousels in DVB-MHP [9]. A single jar file contains the files of the application. Its format is based on the popular zip file [26], which consists of a sequence of file header, file data, and optional data descriptor fields stored sequentially in a single file [27] (Figure 7). The header of each file describes its attributes and parameters needed for extraction, such as CRC32 checksums and compression methods.

To uniquely identify resources, locator strings are also used in Blu-ray. In Blu-ray, the basic identifier is the *disc id*. This identifier is a 128-bit number that uniquely identifies a disc image. Since a disc image contains one or more titles (or programs), each title in the disc image is identified by a title number. The basic Blu-ray locator string is formed by the *disc_id* and the *title number*, as shown below

$$\text{bd://disc\_id.title\_number}, \tag{3}$$

where *disc_id* is a 32-byte string representing the 128-bit *disc id* in hexadecimal notation.

2.3. Related work

To the best of our knowledge, no work has been published on converting iTV interactive data to the Blu-ray format. The closest-related study is presented in [28]. This study analyzes issues involved in the recording of iTV and proposes a method for recording the iTV signal. The proposed method separates the interactive data from the audiovisual content. The interactive data is stored in a tree structure separate from the audio and video streams, in order to save disk space. This method is studied in depth in [29], where a more specific architecture of a system for recording iTV is described. The suggested method and architecture are suitable for enabling personal or digital video recorders (PVRs/DVRs) to record iTV shows. However, our overall goal is to achieve compatibility between DVB-MHP-based broadcast iTV signals and the Blu-ray system, in order to allow real-time playback of the iTV shows in a Blu-ray device. In that respect, the method described in the aforementioned works is not enough, since the format used for storage is not compliant with the Blu-ray system.

An architecture that improves the interactive data downloading process and reduces the iTV application launching time is proposed in [30]. This architecture relies on a cache mechanism for the received DSM-CC *download protocol* messages. It also supports priority-based scheduling of data carousel monitoring threads. The work developed in [30] focuses on the improvement of performance of interactive data reception in iTV receivers.

In [31], we have analyzed some differences between the DVB-MHP and Blu-ray standards and proposed a

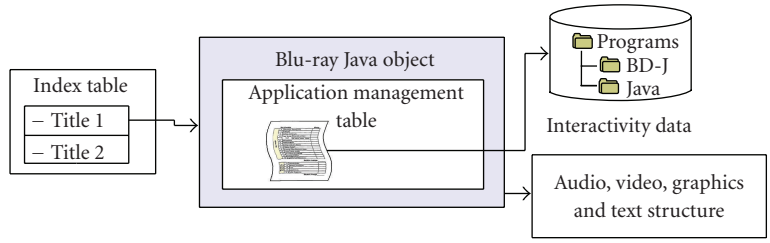

Figure 5: Simplified structure of Blu-ray Java title.

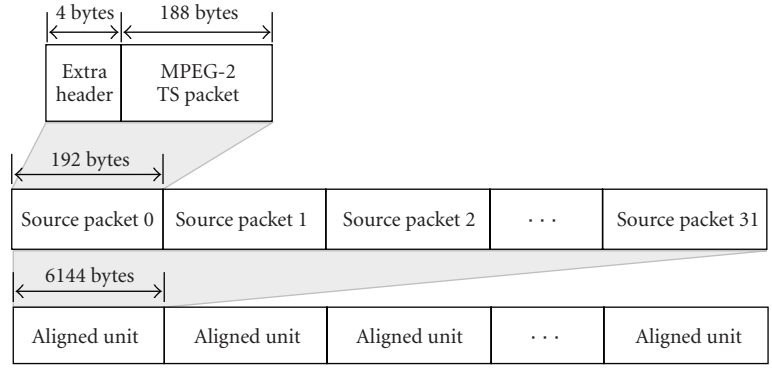

Figure 6: BDAV transport stream.

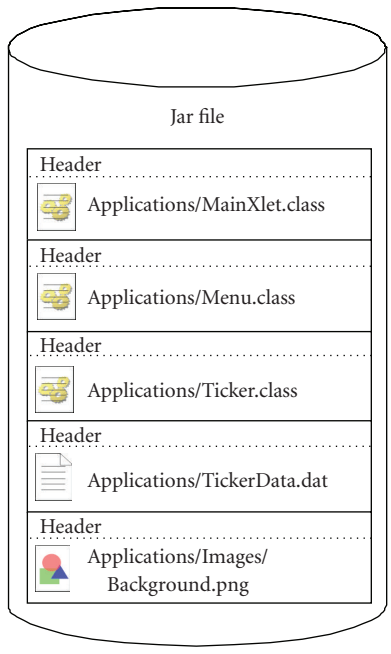

Figure 7: Interactive data format in Blu-ray.

method to transcode DVB-MHP system information to the Blu-ray standard. This method achieves compatibility by modifying the service information tables, adapting them to the constraints imposed by the Blu-ray format. Although system compatibility is the objective of the study in [31], it only addresses some of the problems (i.e., system information) related to the compatibility between the DVB-MHP audiovisual content and the Blu-ray audiovisual streams. However, in order to achieve full compatibility between real-time iTV and Blu-ray, it is also necessary to process the interactive data in the DVB-MHP signal and convert them to compliant Blu-ray format. In this work, we focus on the development of a scheme that converts DVB-MHP interactive data to a Blu-ray compliant format.

3. OUR INTERACTIVE DATA TRANSCODING SCHEME

3.1. Overview of the transcoding scheme

The objective of our interactive data transcoding scheme is to receive the transport stream packets, filter only those sections containing the interactive data and channel metadata, and change the interactive data format to a Blu-ray compliant format. Our transcoder should successfully identify the appropriate sections, separate them into categories that can be processed in parallel and independently of each other, and finally assemble the interactive data that are compliant with the Blu-ray standard. Figure 8 shows a block diagram of our proposed transcoder.

The interactive data contained in the different sections are formatted according to the protocols specified in the extensions for digital storage media command and control (DSM-CC), part of the MPEG-2 standard [18–25]. These sections will be processed by the module transcoder component of our DVB-MHP to Blu-ray transcoder shown in Figure 8. This module will change the DSM-CC formatted interactive data to a Blu-ray compliant format known as a jar file [26]. The channel metadata, known as service

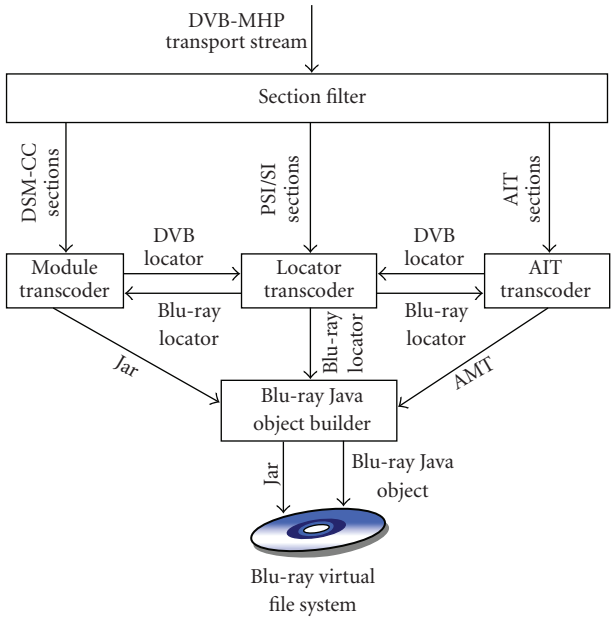

Figure 8: Block diagram of the DVB-MHP to Blu-ray interactive data transcoder.

information (SI) or program specific information (PSI), follow the format specified in the MPEG-2 standard. In DVB-MHP streams, individual channel metadata include the application information table (AIT), which contains information regarding the interactive applications in the channel. The sections carrying the AIT will be processed by the AIT transcoder (see Figure 8), which will generate the corresponding Blu-ray compliant table known as the application management table (AMT). The interactive applications and the AIT may include references to resources in the channel, such as video streams or interactive data files. These resources are identified by special strings known as locators. The locators in the DVB-MHP format must be converted to the Blu-ray format. In DVB-MHP, locators are based on information found in the service information (SI) tables. Our locator transcoder receives SI sections in order to convert DVB-MHP locators to the Blu-ray equivalent format. Finally, to make the transcoded content available to the Blu-ray player, a Blu-ray Java object is built based on the information generated by the different transcoder modules. The information generated will be bound to the virtual file system of the Blu-ray player. In the following sections, we describe the functions of each of the transcoding components and, after analyzing the differences of the interactive information between the two standards, we propose techniques to transcode the DVB-MHP interactive data to a Blu-ray compliant format.

The three main transcoders of our scheme can be launched in parallel, since DSM-CC, PSI/SI, and AIT data are separated from the stream by the section filter and each data category is independent from each other. However, the module transcoder and the AIT transcoder may be dependant on the locator transcoder. This happens if locators are used to identify resources in the interactive data or AIT. This is not a concerning issue since the information required by the locator is transmitted very frequently according to the broadcasting specifications [17].

3.2. Section filter

The section filter component of our transcoder processes the transport stream packets and extracts only the MPEG-2 sections that carry interactive data and channel metadata. These sections, along with audiovisual information, are multiplexed in the transport stream. To perform multiplexing, a multiplexer splits sections into fragments that fit the payload of a transport stream packet, as illustrated in Figure 9. These packets are inserted into the transport stream.

In order to extract the interactive data, the section filter must first build the sections that are carried in the transport stream packets. The header of the transport stream packets contains a *packet id* (see Figure 9). The packets with the same value in the *packet id* carry information of the same elementary stream (for instance, packets with *packet id* = 0014_h carry the audio bitstream). The header also contains a *payload unit start indicator* (*PUSI*) flag (see Figure 9). The value of 1 in this flag indicates that the packet carries the first fragment payload unit (a section or a packetized elementary stream packet). When a transport stream packet is received, the *PUSI* flag is read to check if a new section is beginning in the packet. If the *PUSI* flag is on, then the first three bytes of the payload are read. A value of 000000_h in these bytes indicates that the payload unit started in the packet is a packetized elementary stream (PES) packet. In this case, the packet is discarded, as well as the packets with the same *packet id* value, since they will also contain fragments of PES packets. If the value in the first three bytes of the payload is different than 000000_h, then a buffer is created to store the new section. The payload of the packet is copied to the first bytes of the section buffer, and its *packet id* value is stored in a variable. This value is compared to the packet id of the subsequent packets in the stream. The payload of the packets with the same packet id value is copied after the last bytes copied in the buffer. This operation is performed until the section is complete.

Once a complete section has been formed, the value in the first byte of the section is read. This byte contains the field known as the *table_id* and indicates the type of section; in other words, it indicates whether the section is a DSM-CC section, a PSI/SI section, and\or an AIT section. The *table_id* values for the different types of sections, the type of data, and a brief description of each type are shown in Table 2. Once a section has been categorized by reading the value of the *table_id*, it is sent to one of the different modules of our transcoding system.

3.3. Module transcoder

The task of our module transcoder is to convert the modules containing the DSM-CC formatted files and directories to a single Blu-ray compatible jar file. In order to perform this task, the module transcoder first needs to build the DSM-CC modules (modules 1 and 2 in Figure 3, shown in Section 2).

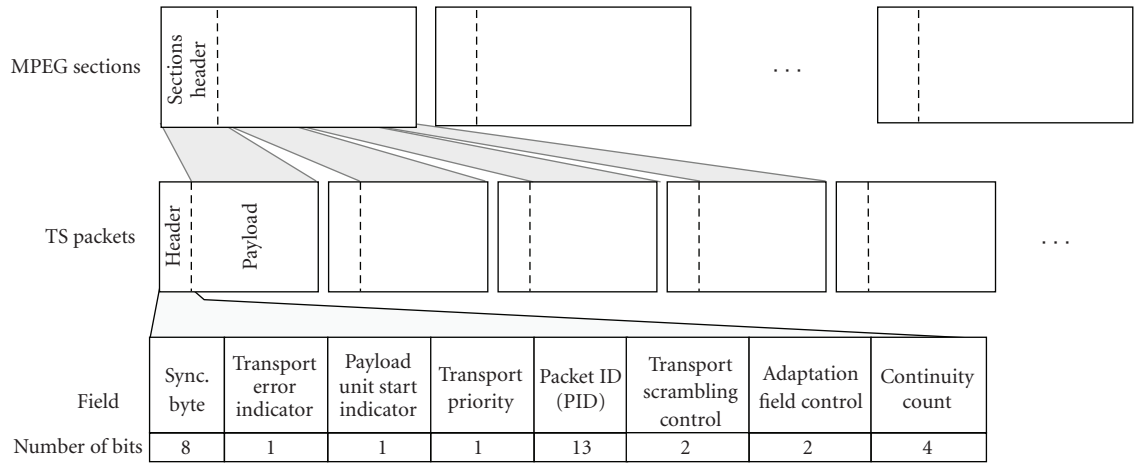

Figure 9: Sections split into transport stream packets.

Table 2: Table_id for different types of sections.

Table_id	Type of data	Description
00_h	PSI/SI	Program management table lists the programs in the stream and associates them with the corresponding program association table
01_h	PSI/SI	Program association table lists the different components of a program (audio, video, and data streams) and associates them with a PID
$3B_h$	DSM-CC	DSM-CC control message. Contains DSM-CC messages for transmission control
$3C_h$	DSM-CC	DSM-CC data messages contains DSM-CC messages for data transmission
74_h	AIT	Application information table describes the interactive applications in the program.

These consist of DSM-CC messages that are transmitted according to the DSM-CC *download protocol* (see Table 1 in Section 2).

In a DVB-MHP stream, a typical sequence of messages begins with a download server initiate (DSI) message. A DSI message notifies the receiver that a specific broadcaster is transmitting information. Following the DSI message, a download info indication (DII) message is transmitted, providing data transfer parameters, such as the number of modules to be transmitted, their IDs, and their sizes. Then, a sequence of download data block (DDB) messages is transmitted. Each DDB message contains a fragment of a module, as depicted in Figure 10. The header of a DDB message contains the id of the module and a block number that indicates the number of the specific data block within the module. Thus, a module M that is split into n data blocks can be built from the sequence of blocks with module id M and block numbers from 1 to n.

The process of constructing the modules from the received DSM-CC messages is known as the download process. DVB-MHP-based iTV receivers start the interactive data download process when they receive a download server initiate (DSI) message. One problem presented by this approach is that the time when a user tunes to a channel may not coincide with the time that the broadcaster sends a DSI message. In this case, the receiver will have to wait until it receives the announcement of an interactive data transfer, discarding all the messages received previous to the DSI message. In order to reduce this waiting time, our module transcoder implements a buffering mechanism in the form of a block cache.

Figure 11 shows a possible sequence received by the receiver when the user tunes to a channel and our block caching buffer. As mentioned before, a set-top box will start the download process only after the first DSI message is received. Therefore, data blocks received previous to a DSI message will be discarded. In our implementation, these blocks will be stored in the block cache, as shown in Figure 11. Since a carousel approach is used for transmitting DVB-MHP interactive data, there is a high probability that the same sequence of data blocks will be transmitted after the DSI message. Therefore, the blocks stored in the block cache can be used to complement the information received after a DSI message. In other words, if the user tunes to a channel when block i of module M is being transmitted, the sequence of blocks with block numbers from i to n is received and stored in the block cache. After the DSI message is received, the download of module M is complete when the blocks numbered from 1 to $i - 1$ have been received, since the remaining set of blocks is already stored in the block cache. The next step is for the module transcoder to assemble the module using this full set of blocks. Since our caching mechanism stores only data blocks, it is simpler

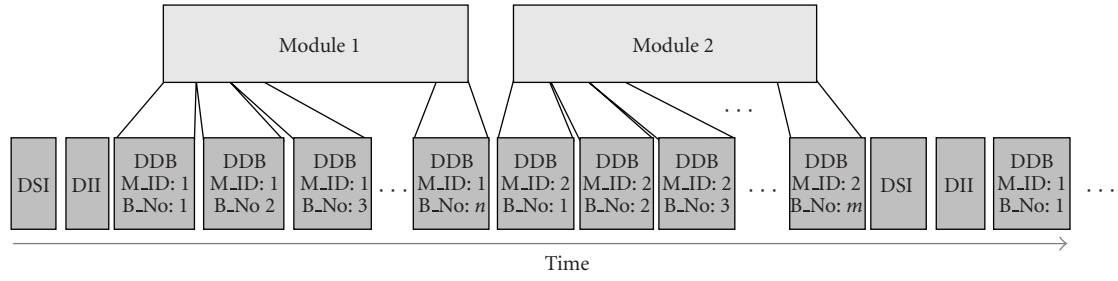

Figure 10: A typical sequence of DSM-CC download messages.

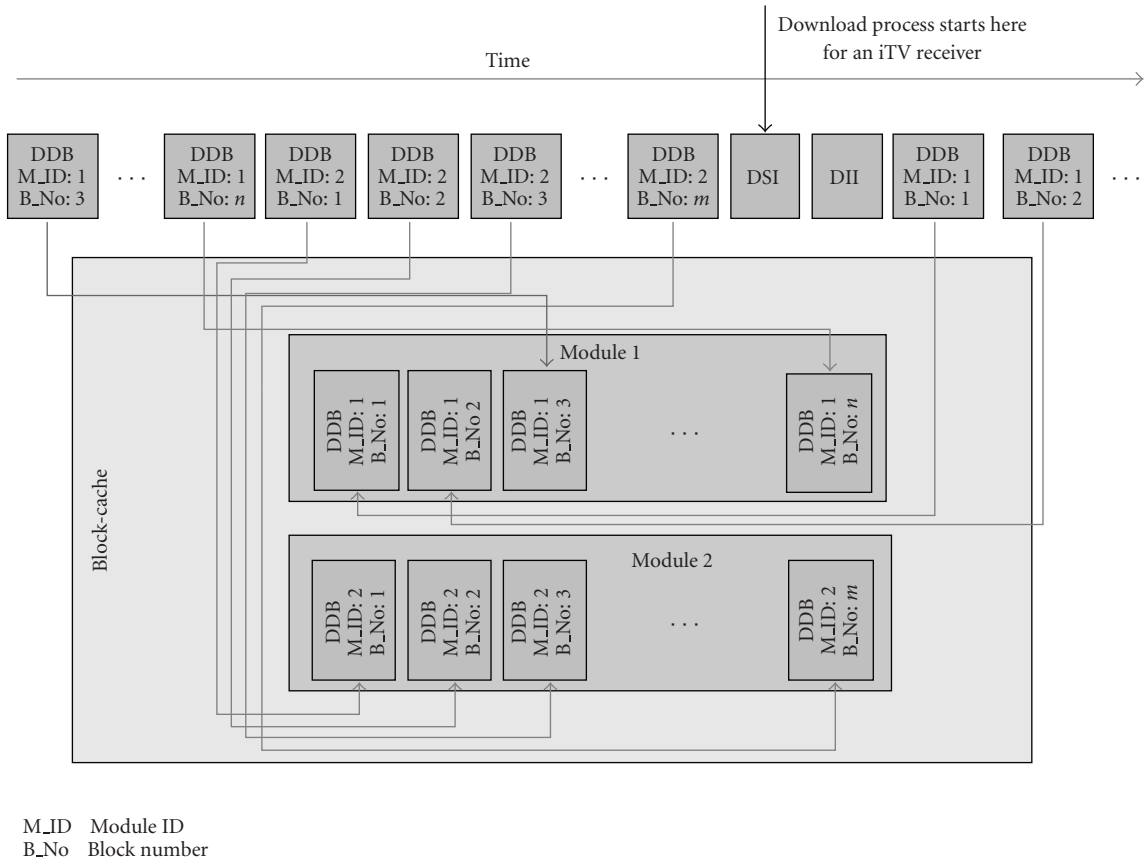

Figure 11: A received sequence of DSM-CC commands and the block cache.

than the caching and monitoring mechanism implemented, presented in [30], which in addition to data blocks stores DSI messages as well.

Once a complete module has been formed, the next stage is to convert the assembled module to the jar file format. According to this format, every file is stored following a header, which contains parameters for retrieving the file (as previously shown in Figure 7, Section 2). For this reason, a jar header is generated by our transcoder for every BIOP message containing a file object, and the body of the message is stored following the header. The jar header (called local file header in the zip file specification) contains a fixed number of fields. Table 3 shows all the fields in the jar header, their size, the description, and the value assigned after transcoding. As can be seen from Table 3, fields 1 to 4 of the header do not depend on DVB-MHP data but rather have fixed values based on the specification [27]. Fields 5 and 6 do not depend on the DVB- MHP data, but on the date of creation or modification of the file stored in the jar. In our transcoding system, when the jar header is created,

Table 3: Fields in the jar header.

	Field	Size (bytes)	Description	Transcoded value
1	Local file header signature	4	Marks the beginning of a new entry	Fixed to $04034b50_h$
2	Version needed to extract	2	Zip specification version needed to extract the file.	Fixed to 0014_h (simple file)
3	General purpose bit flag	2	General purpose flags, bit flags are specified in [27]	Fixed to 0000
4	Compression method	2	Indicates the method used for file compression	Fixed to 0000 (no compression)
5	Last mod file time	2	Last file modification time	Obtained from the system clock
6	Last mod file date	2	Last file modification date	Obtained from the system clock
7	crc_32	4	Cyclic redundancy check code	Based on file data, calculated using a table based fast algorithm
8	Compressed size	4	Size of the file after compression	Copied from the BIOP object headers
9	Uncompressed size	4	Size of the uncompressed file	Copied from the BIOP object headers
10	Filename length	2	Number of bytes in the filename	Obtained after the filename is built
11	Extra field length	2	Length of the extra field	Fixed to 00 (no extra field)
12	Filename	Variable	Filename including the path (directories)	Generated from objects in the carousel
13	Extra field	Variable	User defined extra fields	Not included in the generated header

the date and time are obtained from the system clock and stored in these fields. The values for fields 8 and 9 in the jar header are obtained directly from the BIOP message headers. The size of a file in DBV-MHP is specified in the headers of the corresponding BIOP file object. In Blu-ray and the corresponding jar header, this value is stored in *uncompressed size* (field 8) and since no compression is applied to the file, the value is also stored in *compressed size* (field 9).

The value for field 7 in the jar header must be generated based on the received information. This field contains an error-protection code based on the cyclic redundancy check (CRC) function known as *crc_32*, which is calculated using a 32-degree polynomial. In the implementation of our transcoder, a fast algorithm that uses precalculated tables to accelerate the polynomial evaluation is used. The field in the jar header that is the most difficult to obtain from the corresponding DVB-MHP content is the filename (field 12). In addition to the name of the file, the filename field must include the path to the file (in other words, its location in the directory structure). In the object carousel, the filename for a specific BIOP file object cannot be obtained directly from the BIOP message headers. It must be constructed from information contained in other objects in the carousel (Figure 12).

As mentioned before, the object carousel is formed by directory and file objects. Every object in the carousel is uniquely identified by an *object key*. Both the object type and the *object key* are fields of the BIOP message header. The body of a message containing a file object is the actual data of the file, and the size or length of the file is stored in the BIOP message subheader. The body of a message for a directory object is a list of the objects in the directory (files and subdirectories in a regular directory structure). Every entry in this list is a link to the actual objects and is described by the name (filename or subdirectory name) of the linked object; its object type and object key (see Figure 13). The number of links (also called *bindings*) in a directory object is stored in its BIOP message subheader. Thus, for each received BIOP file object, the name and path must be traced back in the tree formed by the linked BIOP objects in order to generate the corresponding filename field for the jar header. For instance, in the example shown in Figure 12, to obtain the filename of object (e), the name must be obtained from the parent directory object (d) and the path must be traced from object (d) to the root directory object (a) (also called service gateway). If any of the parent objects of a DVB-MHP file object has not been received when its corresponding jar header is created, the filename required in the jar header cannot be generated. For instance, in the example shown in Figure 12, if the header for object (e) is created but object (d) has not been received, the full filename cannot be obtained, since object (d) contains the filename and the link of the file object to the rest of the tree structure. In our transcoder, if the object to be transcoded is not linked to any other received object, the jar entry (jar header followed by the file content) with the header partially filled is stored in an associative array (as shown in Figure 13), where the array element key (index) is the object key. In this case, the object is recovered later, when the parent objects are received, and the filename can be obtained.

All the received BIOP messages containing directory objects are stored in a separate array. When a new directory object is received, it is stored in this array. Then, the object keys in the directory objects are searched to verify if any of the partially filled jar entries can be completed. If the object key of a partially filled jar entry is found, the tree formed by

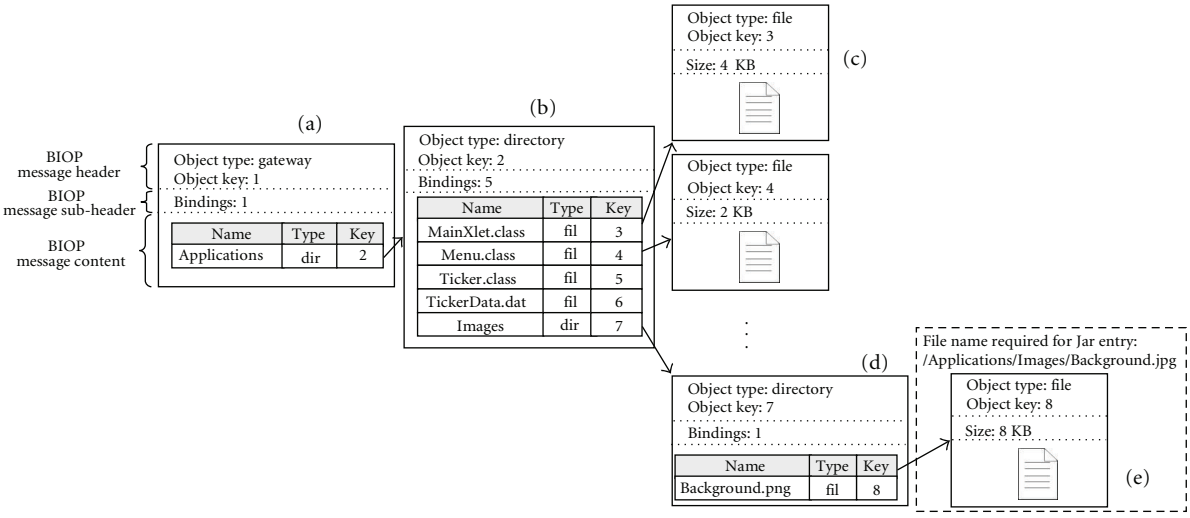

Figure 12: Organization of the example BIOP objects.

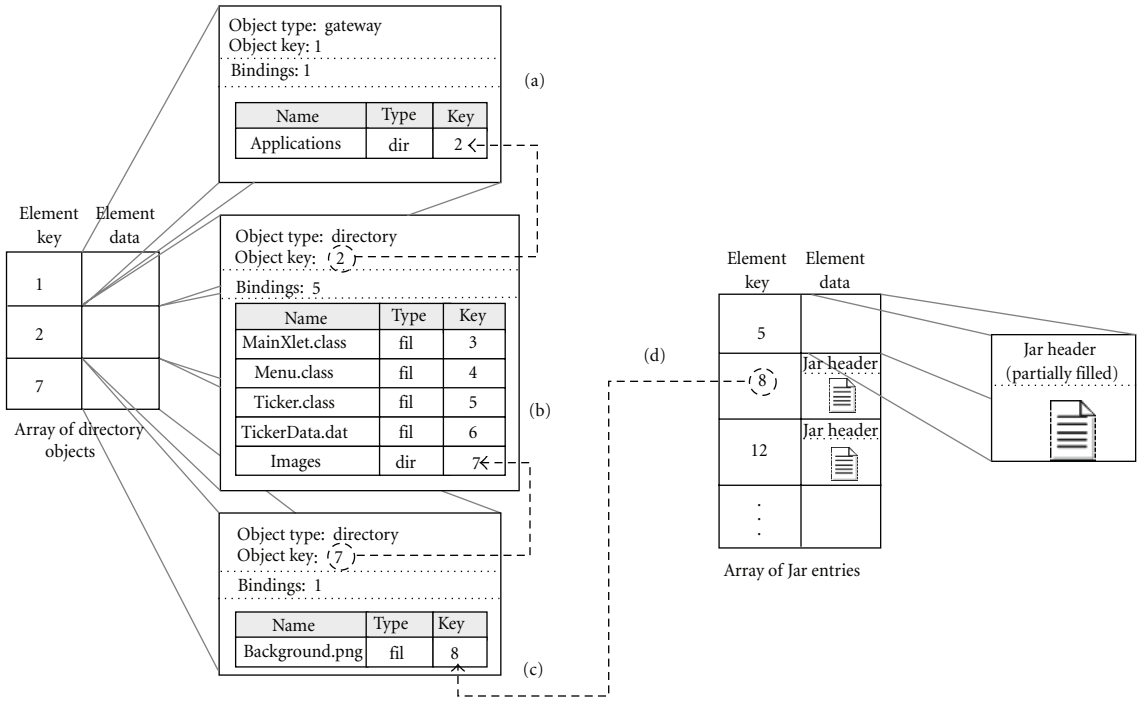

Figure 13: Arrays of jar entries and directory objects, with the route for tracing the filename for a jar entry.

the directory objects is traced back to build the full filename required for the jar entry header (Figure 13). When the full filename of an object can be built based on the directory objects in the array, the header of the corresponding jar entry is completed, and the entry is placed in the jar file. In the example shown in Figure 12, when object (c) arrives and is stored in the array of directory object messages, the object keys in the bindings list of the object can be searched in the array of jar entries. There is only one key in the bindings list of (c). When this key is found as the key for element (d), the object key of (c) is searched in the remaining elements in the array directory object messages (objects (a) and (b)). The object key of (c) will be found in the bindings list of (b), and then the object key of (b) will be searched in the remaining object of the array. This step is recursive and is repeated until the root object of the tree (object (a)) is reached. If a key is not found, then the path of the file cannot be determined, and the process will be repeated when a new directory object is received.

3.4. AIT transcoder

This part of our transcoder processes the MPEG-2 sections that carry the DVB-MHP application information

table (AIT), and converts them to the equivalent Blu-ray application management table (AMT). The AIT is a service information table included in DVB-MHP, and provides information about the interactive applications being broadcasted and actions related to their execution state. For instance, it contains flags that are used to indicate whether an application must be executed immediately, terminated, or executed when a button on the remote control is pressed.

The AIT is carried in the MPEG-2 sections and follows the standard MPEG-2 service information table (SIT) format. According to the MPEG-2 specification, a service information table (SIT) consists of basic fields and descriptors. In total, there are 10 different types of descriptors. The use of some specific descriptors in an AIT table is mandatory, while other descriptors and the associated fields are optional. All the basic fields are mandatory. The Blu-ray equivalent to the DVB-MHP AIT is the application management table (AMT). The AMT has a smaller number of basic fields than an AIT, and only one type of descriptor. The fields grouped under this descriptor are equivalent to fields grouped in several descriptors of the DVB-MHP AIT.

Figure 14 shows the DVB-MHP AIT and the Blu-ray AMT, as well as their fields and descriptors. The AIT in Figure 9(a) represents one of many possible combinations of fields showing only 5 of the 10 possible descriptors. The position of these descriptors in the table is not fixed; in other words, fields can be in different positions within the table. In both tables, there are blocks of fields that can be repeated several times. In Figure 14, such blocks are marked by letters. The number of repetitions is usually determined by the field located just before the block of fields. In the Blu-ray AMT, the size (in bytes) of each of these blocks of fields must be a multiple of 2, since the data unit in the Blu-ray read system is word-based. For this reason, additional empty fields, known as word alignment fields, are added after each block of fields in the AMT, as shown in Figure 14(b).

Our approach for transcoding a DVB-MHP AIT to a Blu-ray AMT is to create AMT fields on a block-by-block basis. The values of the grey fields in Figure 14(b) are either transcoded from the corresponding DVB-MHP or are generated according to the Blu-ray specification. If an AMT field does not have a corresponding field in AIT (usually because the descriptor is not present), then it is assigned a default value-based on the Blu-ray specification. The first AMT fields are the basic fields *length* and *number_of_applications*. The *length* field indicates the size of the table in bytes. This value is assigned once the creation of the table has been completed, since it contains the total number of bytes needed to store the table. The *number_of_applications* field indicates the number of applications that are going to be described in the table. This field is equivalent to the *application_loop_length* field of the AIT (field 16 of Figure 14(a)). However, the equivalent field in AIT field indicates the number of bytes used for describing the applications, instead of the number of applications. It is not possible to calculate the number of applications based only on the number of bytes used, since the use of some descriptors is optional and the number of fields used to describe each of the applications is not constant. In order to calculate the value for the *number_of_applications*, a counter is used. This counter is initialized to 0 and incremented when a block of AIT fields describing a single application is read (block a in Figure 14(a)). The value for *application_control_code* is generated as follows. In Blu-ray, this field can have values that are restricted to a set smaller than the set of values allowed in DVB-MHP. If in the AIT the value in this field is greater than the highest allowed in Blu-ray, the value assigned in the AMT is the highest value allowed; otherwise, the value assigned is a copy of the value in AIT. The AMT *application_type* field indicates the type of application that is described in block a of Figure 14(b). In the current specification of Blu-ray, only one type of application is supported, known as Blu-ray Java. For this reason, the value of this field is always 1, which is the identifier for the Blu-ray Java application. However, since DVB-MHP supports more than one application type, it is necessary to check that the value of the corresponding AIT field is equivalent to the Blu-ray Java application. If the value in field 7 in Figure 14(a) is different from 1 (DVB-Java), then the transcoding process must be stopped since no Blu-ray compatible application is being transmitted in the stream. As for fields 6 and 7 of the AMT, they give the id of the organization that provides the interactive application (assigned by the DVB group) and the id of the application (assigned by the organization). Their values are directly copied from the equivalent AIT fields (fields 17 and 18 in Figure 9(a)). The value for the AMT *descriptor_tag* is fixed to 00_h, which is the value required by Blu-ray. The value for *descriptor_length* is obtained once the complete block of fields, labelled as a in Figure 14(b), has been created, since it indicates the number of bytes used to store the entire block. The *application_profiles_count* in the AMT indicates the number of application profiles described in the table. An application profile is a description of a series of minimum configurations, providing different capabilities of the MHP [1]. In other words, it specifies the minimum functions in the receiver or the player required by the interactive application. Two different profiles and their respective capabilities are defined in the DVB-MHP specification and are the same as defined in the Blu-ray specification. The equivalent field in the AIT is *applications_profile_length*. This field gives the number of bytes used to specify all the application profiles in the table. In the AIT, 5 bytes are used for storing a single application profile (block b in Figure 9(a)). Therefore, in order to calculate the number of application profiles, which is the value for the AMT *application_profiles_count*, the value of the AIT *application_profiles_length* must be divided by 5. Once this step is completed, the following fields, shown as block b in AMT, are directly copied from the corresponding fields in AIT (see Figure 14). The problem here is that the generated AMT block b consists of 5 bytes, and word alignment is needed. For this reason, an extra empty field 1-byte long is added to this block (see Figure 14(b)).

The AMT fields *application_priority, application_binding,* and *visibility* indicate the relative priority between the applications, if an application must be terminated when the corresponding title is changed and the visibility to the end user, respectively. The values of these fields are directly copied

DVB-MHP application information table

		Syntax	Size (bits)	Descriptor
	1	Table_id	8	
	2	Section_syntax_indicator	1	
	3	Reserved_future_use	1	
	4	Reserved	2	
	5	Section_length	12	
	6	Test_application_flag	1	
	7	Application_type	15	
	8	Reserved	2	
	9	Version_number	5	
	10	Current_next_indicator	1	
	11	Section_number	8	
	12	Last_section_number	8	
	13	Reserved_future_use	4	
	14	Common_descriptors_length	12	
	15	Reserved_future_use	4	
	16	Application_loop_length	12	
a	17	Organisation_id	32	
a	18	Application_id	16	
a	19	Application_control_code	8	
a	20	Aeserved_future_use	4	
a	21	Application_descriptors_loop_length	12	
a	22	Descriptor_tag	8	Application descriptor
a	23	Descriptor_length	8	
a	24	Application_profiles_length	8	
a b	25	Application_profile	16	
a b	26	Version.major	8	
a b	27	Version.minor	8	
a b	28	Version.micro	8	
a	29	Service_bound_flag	1	
a	30	Visibility	2	
a	31	Reserved_future_use	5	
a	32	Application_priority	8	
a c	33	Transport_protocol_label	8	
a	34	Descriptor_tag	8	Data broadcast id descriptor
a	35	Descriptor_length	8	
a	36	Data_broadcast_id	16	
a d	37	Application_type	16	
a	38	Descriptor_tag	8	Application name descriptor
a	39	Descriptor_length	8	
a e	40	ISO_639_language_code	24	
a e	41	Application_name_length	8	
a e f	42	Application_name_char	8	
a	43	Descriptor_tag	8	DVB-J. application location descriptor
a	44	Descriptor_length	8	
a	45	Base_directory_length	8	
a g	46	Base_directory_byte	8	
a	47	Classpath_extension_length	8	
a h	48	Classpath_extension_byte	8	
a i	49	Initial_class_byte	8	
a	50	Descriptor_tag	8	DVB-J. application descriptor
a	51	Descriptor_length	8	
a j	52	Parameter_length	8	
a j k	53	Parameter_byte	8	
	54	CRC_32	32	

(a)

Blu-ray application management table

		Syntax	Size (bits)	Descriptor
	1	Length	32	
	2	Number_of_applications	8	
	3	Reserved_for_word_align	8	
	3	Application_control_code	8	
	4	Application_type	4	
	5	Reserved_for_word_align	8	
	6	Organization_id	32	
	7	Application_id	16	
	8	Descriptor_tag	8	
	9	Reserved_for_word_align	8	
	10	Descriptor_length	32	
	11	Reserved_for_future_use	32	
	12	Application_profiles_count	4	
	13	Reserved_for_word_align	8	
b	14	Application_profile	16	
b	15	Version.major	8	
b	16	Version.minor	8	
b	17	Version.micro	8	
b	18	Reserved_for_word_align	8	
	19	Application_priority	8	
	20	Application_binding	2	
	21	Visibility	2	
	22	Reserved_for_word_align	8	
a	23	Number_of_application_name_bytes	16	
a c	24	Application_language_code	24	
a c	25	Application_name_length	8	Application descriptor
a c d	26	Application_name_byte	8	
a e	27	Reserved_for_word_align	8	
	28	Application_icon_locator_length	8	
a f	29	Application_icon_locator_byte	8	
a g	30	Reserved_for_word_align	8	
	31	Application_icon_flags	16	
	32	Base_directory_length	8	
a h	33	Base_directory_byte	8	
a i	34	Reserved_for_word_align	8	
	35	Classpath_extension_length	8	
a j	36	Classpath_extension_byte	8	
a k	37	Reserved_for_word_align	8	
	38	Initial_class_name_length	8	
a l	39	Initial_class_name_byte	8	
a m	40	Reserved_for_word_align	8	
	41	Number_of_overall_parameter_bytes	8	
a n	42	Number_of_parameter_bytes	8	
a n o	43	Parameter_byte	8	
a p	44	Reserved_for_word_align	8	

(b)

Figure 14: Application information table (AIT) (a) and application management table (AMT) (b).

from the equivalent AIT fields (fields 32, 29, and 30, resp., in Figure 14(a)). The values of field 23 and field in the block labelled as c in Figure 14(b) are also directly copied from the AIT equivalent fields (field 39 and block e in Figure 14(a)). These fields are used for providing the name and language of the application. Block f in the AMT (see Figure 14(b)) gives the filename of the image used as an icon for the application, and the field *application_icon_locator_length* indicates the length of this filename. In the AIT, shown in Figure 14(a), there is no equivalent to these fields. For this reason, the AMT block f is not generated, and the value 0 is assigned to *application_icon_locator_length*.

The fields of block h in the AMT contain the name of the initial directory of the application. Since an application in Blu-ray is stored in a jar file, this field must begin with the name of the jar, as required in the Blu-ray specification. The equivalent to this block in the AIT is the block labelled g in Figure 14(a). This block is copied after the name of the jar file (which was generated by the module transcoder) in AMT block h. The *base_directory_length* contains the number of bytes used for storing the base directory block. This field can be calculated by adding 5 (the length of the jar filename) to the value of the corresponding field in the AIT (field 45 in Figure 14(a)).

The block of *class_path_extension_byte* fields gives the path to the directory containing libraries used by the application. The value of this field in the AMT and the field indicating the size in bytes of the block

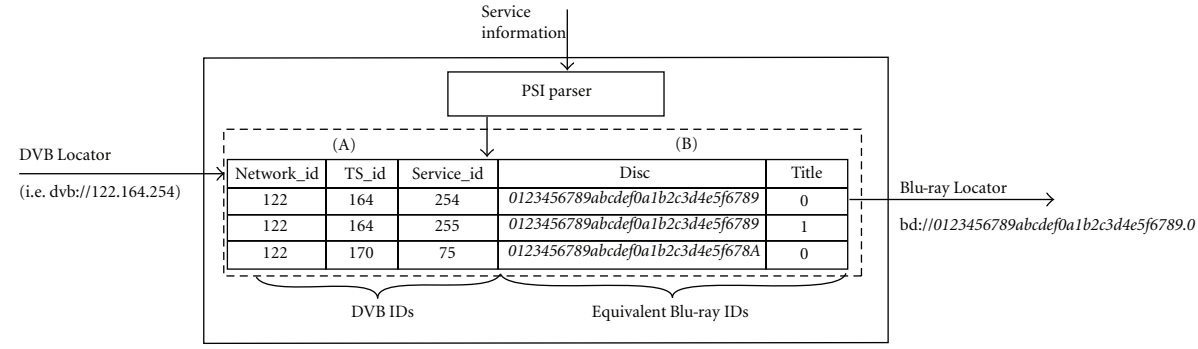

Figure 15: Locator transcoder mapping table.

(field 35 in Figure 14) are directly copied from their equivalent fields in AIT (fields 47 and 48 in Figure 14(a)). The block of *initial_class_name_byte* fields provides the file in the application that is used as the starting point in its execution. This block in the AMT is directly copied from its equivalent in the AIT (labelled d in Figure 14(b) and i in Figure 14(a)). In the AMT, the length of this block must be stored in field 38 (see Figure 14). This field does not exist in the AIT, but can be calculated, since the *descriptor_length* (field 44 of AIT) contains the total number of bytes from field 45 to field 49.

The last block of AMT fields is the block n shown in Figure 14(b), which contains parameters for the application. This block is directly copied from the AIT equivalent (block j in Figure 14(a)). AMT field 41 shows the number of times this block is repeated. In order to calculate its value, a counter is used that is incremented every time the block is read.

3.5. Locator transcoder

The locator transcoder converts the DVB-MHP *locators* to the Blu-ray *locator* format. The basic task of this transcoder is to map the DVB specific identifiers to the unique identifier of the disc image (*disc_id)* in order to create Blu-ray locators that are equivalent to *DVB-MHP locators.* The *disc_id* is assigned by the content provider. In DVB-MHP, the *transport stream id* is also assigned by the content provider. For this reason, in our transcoder, the pair formed by a *network id* and a *transport stream id* identifiers is equivalent to a Blu-ray *disc_id*. Thus, when a new pair of *network id* and transport *stream id* is read from the service information tables, a *disc_id* is created. For the creation of such a *disc id*, a disc counter is stored in the transcoder system. This counter is incremented every time a *disc_id* is created, therefore, every *disc_id* will have a different value. A disc image contains several titles that are listed in the index table of the disc image in Blu-ray. Each title is addressed by a *title number*, which is directly equivalent to its position in the index table. Similarly, in DVB-MHP, a transport stream can contain several programs. The list of these programs can be found in the service information tables and can be seen as the equivalent to the index table; therefore, the position of a program in the programs list is equivalent to the Blu-ray title number. When the network id and transport *stream id* are read from the service information, the locator transcoder also reads the list of programs in the transport stream and the identifier of each program (*service id*). The three identifiers are stored in a table. Each row in the table contains a unique combination of the values of *network id*, *transport stream id,* and *service id* found in the service information, as shown in Figure 15(a). In this table, the Blu-ray *disc_id* and *title_number* equivalent to the DVB-MHP identifiers are stored in each row (Figure 15(b)). This table is used for fast mapping of the DVB locator to Blu-ray resources. When a DVB locator is received by the locator transcoder, it reads the different ids in the locator and looks for the row that has the same values in the corresponding IDs. Once the row is found, the equivalent Blu-ray identifiers are read, and a locator in the format required by Blu-ray is formed.

Table 4: Sections of the Blu-ray Java object.

Blu-ray Java object section	Description
Terminal information	Contains flags related to title changes and configuration data.
Cache information	This section contains information indicating the files that must be cached in order to improve disk performance.
Table of accessible play lists	Contains information related to the access control of audiovisual information.
Application M management table	Provides information related to the applications associated to the title.
Key interest table	Contains information related to playback control of user generated events (flags for enabling or disabling input from buttons related to playback).
File access information	Determines read access to the files in the virtual file system of the player.

3.6. Blu-ray Java object builder

The Blu-ray Java object indicates to the player what audiovisual content and interactive data belong to a title. It also

Table 5: Terminal information.

	Syntax	Size (bits)
1	Length	32
2	Default_font_filename	8∗5
3	Initial_HAVi_configuration_id	4
4	Menu_call_mask	1
5	Title_search_mask	1
6	Reserved_for_future_use	34

Table 6: Application cache information.

		Syntax	Size (bits)
	1	Length	32
	2	Number_of_entries	8
	3	Reserved_for_word_align	8
a	4	Entry_type	8
	5	Ref_to_name	8∗5
	6	Language_code	8∗3
	7	Reserved_for_future_use	8

contains information about the functions in the player that can be used for the specific title. When a user selects a title, the player first reads the Java object and then looks for the information about the content associated with the title. The Blu-ray Java object is a unique structure, not present in DVB-MHP. Our task is to build a Blu-ray Java object for each title.

According to the Blu-ray specification, the Blu-ray Java object consists of six sections. The different sections of the Blu-ray Java object and a brief description of each of them are listed in Table 4.

The first sections in the object are the *terminal information* and the *cache information*. Tables 5 and 6 show the fields in these sections and the size of each field. Table 5 contains information about functions that will be available in the player while the title is being played. The functions permitted when broadcast content is played will always be the same; therefore, the values for the fields in this table are fixed. The *cache information* table (Table 6) provides information about jar files to be stored in the local storage of the player in order to improve performance. This caching approach is good for prerecorded media, where the application files never change. This approach is not suitable for broadcast media; therefore, no applications will be cached. The fields of block a in Table 6 are not created, and the values for the remaining fields are fixed.

Table 7 shows the *accessible playlists*, which link the audiovisual streams that are going to be played in the title. In Blu-ray, in order to allow audiovisual content navigation, the content is split in several streams, each one containing one section of the audiovisual material (e.g., each stream can contain a scene in a movie). In contrast to Blu-ray, in broadcast iTV, the content is always played in the same sequence that is received and is not split. Therefore, in this table only, one playlist must be considered. The values for most of the fields (see Table 7) are fixed, since they

Table 7: Table of accessible playlists.

		Syntax	Size (bits)
	1	Length	32
	2	number_of_acc_PlayLists	11
	3	access_to_all_flag	1
	4	autostart_first_PlayList_flag	1
	5	reserved_for_future_use	19
A	6	PlayList_file_name	8∗5
	7	Reserved_for_word_align	8

Table 8: Key interest table.

	Syntax	Size (bits)
1	VK_PLAY	1
2	VK_STOP	1
3	VK_FAST_FWD	1
4	VK_REWIND	1
5	VK_TRACK_NEXT	1
6	VK_TRACK_PREV	1
7	VK_PAUSE	1
8	VK_STILL_OFF	1
9	VK_SECONDARY_AUDIO_ENABLE_DISABLE	1
10	VK_SECONDARY_VIDEO_ENABLE_DISABLE	1
11	VK_PG_TEXTS_ENABLE_DISABLE	1
12	Reserved_for_future_use	21

describe the parameters necessary to read only one playlist, and the only field that is filled with a variable value is *play_list_filename*. This field gives the name of the playlist associated to the title. This name will be the same as the title number.

The following section in the Blu-ray Java object is the application management table (AMT), which has been previously built by the AIT transcoder. The AMT that is received from the AIT transcoder by the object builder is copied after the table of accessible playlists. Once the AMT has been copied to the Blu-ray Java object, the key interest table (Table 8) is created. This table provides information about the remote control buttons that will be enabled during the title playback. Some of these buttons are not useful for broadcast signal playback. For instance, fast forwarding and rewind are not possible during playback of a broadcast signal. All the fields in the table will be set to 0 (button disabled) except for fields 1, 2, 9, and 10 (play, stop, secondary audio, and video, resp.).

The last section is the file access info section. This section contains only 2 fields: the *directory_paths_length* (16 bits), which contains the size of the path string, and *directory_paths_byte* (8 bits, repeated *directory_paths_length* times), which is a block of bytes that contains a path string. The path string contains the path to the directory, where the jar files are stored, since the player needs to access the files of the Java application.

Table 9: Test results of the interactive data transcoder.

Stream		Stream 1 – Generated using available software – Single program multiplex	Stream 2 – Source: DVB-S network – 19 program multiplex (10 audio only)	Stream 3 – Source: from a DVB-T channel – Single program multiplex
Stream data rate (Mbps)		8	38	14
Interactive application data rate (Mbps)		3.09	0.244	7.12
Interactive application size (bytes)		41 498	194 253	1 545 513
Number of modules		2	21	37
Average time for building and transcoding a module (seconds)		0.007	0.054	0.068
Total transcoding time straightforward download	Carousel loops	1	1	2.02
	Seconds	0.0253	1.1754	3.1316
Total transcoding time cache-based download	Carousel loops	0.4920	0.8446	0.0203
	Seconds	0.0043	0.9692	1.0342
Time improvement by using cache	Carousel loops	0.508	0.1554	1.997
	Seconds	0.0210	0.2062	2.0974

4. PERFORMANCE EVALUATION AND DISCUSSION

An implementation of our interactive data transcoder was tested using three different iTV streams. One of the streams was generated by encoding a sample iTV application into an object carousel and multiplexing it along with a test audiovisual stream. The encoding of applications and multiplexing of the encoded interactive data with the audiovisual stream were done using tools made available by the open source community [32]. The second and third streams were obtained from the iTV development community. The original source of the second stream is a DVB-satellite channel belonging to the Astra network. The third stream belongs to the Cineca Inter-university Consortium and the source of this stream is a DVB-terrestrial channel.

The characteristics of the test streams and the corresponding transcoding time are presented in Table 9. The first row of the table gives a brief description of the streams, including the source and number of programs in each stream. The second, third, fourth, fifth, and sixth rows show the stream data rate, the data rate of the interactive application, the size of the interactive application, the number of modules used to broadcast the application, and the average time used for building and transcoding a module, respectively. The last six rows of the table show the improvement of the transcoding process when the data block cache was used. The seventh and eighth rows show the total transcoding time using a straightforward interactive data download approach. The ninth and tenth rows show the total transcoding time using the data block cache. The eleventh and twelfth rows show the time saved by our data block caching approach.

We applied the transcoding to full sets of buffered TS packets. Since each stream was already buffered in a single file, no transmission delays affected the transcoding time measurement process. Transcoding time was measured on a personal computer and spans from the reception of the first packet containing interactive data to the completion of the transcoding process. In the results where the carousel loops are used as a measuring unit, we considered that a DSI/DII message indicates the beginning of a loop. The carousel loop spans from the reception of the DSI/DII message to the last DDB message of the carousel. The first DSI/DII indicates the beginning of first carousel loop.

We observe that the average transcoding time for a module for all three streams is very small, allowing for real-time playback. The difference in times is proportionally related to the data rate of the interactive stream, which affects the time for constructing a module, and the complexity of the interactivity involved. As we can see, the second stream takes a bit longer but this is directly related to the data rate which is much slower. Although the third stream is faster than the first one, it requires a bit more time than the first due to the complexity of the interactivity and the fact that many packets were missing from this stream due to transmission errors. Many of these packets were part of sections containing interactive data, and for this reason, the building process for some modules took longer. However, the overall transcoding process was improved by using the proposed block cache mechanism, as it can also be observed in Table 9. The block caching mechanism allowed us to complete the transcoding process before the first full carousel loop was received. In carousel loops, the total transcoding time of the first stream was improved by 50% when the block cache mechanism

was used. The transcoding time of the second stream was improved by 16% and the transcoding time of the third stream by 200%.

One concern about the block caching mechanism is the memory usage. The worst case scenario for memory usage in the block cache occurs when the iTV channel is tuned in right after a DSI/DII message has been transmitted. In this case, the complete set of DDB messages is received and stored in the cache, and the memory used by the block caching mechanism is equivalent to the size of the application. An average iTV application does not exceed 1 MB in size. Large applications grow up to 2 MB. This size is low compared to the minimum amount of memory required by the Blu-ray specification (256 MB [9]). Furthermore, the whole transcoding process is performed using Blu-ray local storage. When data in the block cache are transcoded, the transcoded data are then stored in the same storage unit. The memory used by the caching mechanism is equivalent to the memory used by transcoded data. Therefore, the memory usage in the whole transcoding scheme remains constant.

The results of our transcoding were tested by successfully playing back the interactive streams using a Blu-ray player.

5. CONCLUSIONS

We have developed a mechanism for transcoding a DVB-MHP interactive data stream to a Blu-ray compliant format. This is the first published work on converting iTV interactive data to Blu-ray. An interactive data downloading process is necessary in order to perform the transcoding of the interactive application. A simple cache mechanism that improves the efficiency of this process is proposed. This mechanism takes advantage of the interactive information received before the arrival of the corresponding transmission parameters and is simpler than the mechanism proposed in previous works. Performance evaluations show that our cache mechanism improved the performance by up to 200%.

The downloaded DSM-CC object carousel is converted to the Blu-ray compliant jar file format. In order to accelerate the conversion process, an associative array of DSM-CC directory objects and an associative array of partially converted DSM-CC file objects are used. The use of these arrays allows the conversion of file objects prior to the arrival of the objects that describe the directory structure. The metadata tables that describe the applications and their corresponding parameters are also transcoded. In order for the Blu-ray system to be able to access the transcoded content, the appropriate Blu-ray metadata files are generated.

Although only interactive data were transcoded by our scheme, our results prove that compatibility between broadcast iTV and the Blu-ray system is possible, since the content originally available in the iTV stream has been made available to the Blu-ray system. For a complete DVB-MHP to Blu-ray transcoding system, our future work will focus on the development of a full conversion scheme for audiovisual content.

Making DVB-MHP iTV content compatible with the Blu-ray format and allowing Blu-ray devices to playback the most widely used open standard for iTV not only reduce complexity, but they are also cost effective for manufacturers and consumers. Manufacturers can benefit by offering a device capable of both prerecorded media and iTV playback, and users will appreciate the resulting simplicity and cost savings.

ACKNOWLEDGMENTS

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada. The author would like to thank the Mexican Science and Technology National Council (CONACyT) and the Universidad Autónoma de Baja California for their support.

REFERENCES

[1] "DVB Multimedia Home Platform (MHP) Specification 1.0.3," European Telecommunications Standards Institute TS 102 812, 2003.

[2] J. Piesing, "The DVB multimedia home platform (MHP) and related specifications," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 237–247, 2006.

[3] K. Arnold and J. Gosling, *The Java Programming Language*, Addison-Wesley, Reading, Mass, USA, 2nd edition, 1997.

[4] J. Gosling and H. McGilton, *The Java Language Environment*, Sun Microsystems, Mountain View, Calif, USA, 1996.

[5] A. Lugmar, S. Niiranen, and S. Kalli, *Digital Interactive TV and Metadata: Future Broadcast Multimedia*, Springer, Berlin, Germany, 2004.

[6] European Broadcasting Union, Digital Video Broadcasting Group, "GEM 1.0.2: Digital Video Broadcasting (DVB); Globally Executable MHP (GEM)," European Telecommunications Standards Institute TS 102 819 V1.3.1, 2005.

[7] Society of Cable Telecommunications Engineers, "SCTE Application Platform Standard OCAP 1.0 profile," SCTE 90-1, 2005.

[8] Advanced Television Systems Committee, "ATSC Standard: Advanced Common Applications Platform (ACAP)," ATSC A/101, 2005.

[9] Blu-Ray Disc Association, "System Description Blu-Ray Disc Read-Only Format, Part 3: Audio Visual Basic Specifications," 2006.

[10] S. Morris and A. Smith-Chaigneau, *Interactive TV Standards: A Guide to MHP, OCAP and Java TV*, Focal Press, Oxford, UK, 2005.

[11] U. H. Reimers, "DVB—the family of international standards for digital video broadcasting," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 173–182, 2006.

[12] Moving Picture Experts Group, "Information technology—generic coding of moving pictures and associated audio information: video," ISO/IEC 13818-2:2000, 2000.

[13] Moving Picture Experts Group, "Information technology—generic coding of moving pictures and associated audio information: audio," ISO/IEC 13818-3:2000, 2000.

[14] Moving Picture Experts Group, "Information technology—Generic coding of moving pictures and associated audio information: systems," ISO/IEC 13818-1:2000, 2000.

[15] European Telecommunications Standards Institute, "Implementation guidelines for the use of video and audio coding in broadcasting applications based on the MPEG-2 transport stream," Tech. Rep. TS 101 154, ETSI, Cedex, France, June 2005.

[16] European Telecommunications Standards Institute, "Specification for the use of video and audio coding in DVB services delivered over IP," Tech. Rep. TS 102 005, ETSI, Cedex, France, July 2007.

[17] Digital Video Broadcasting, "Specification for Service Information (SI) in DVB systems," European norm.

[18] Moving Picture Experts Group, "Generic coding of moving pictures and associated audio information—part 6: extension for digital storage media command and control (DSM-CC)," International Standards Organization/International Electrotechnical Comission 13818-6, 1997.

[19] European Telecommunications Standards Institute, "Implementation guidelines for Data Broadcasting," Tech. Rep. TR 101 202, ETSI, Cedex, France, 2003.

[20] V. Balabanian, L. Casey, N. Greene, and N. C. Adams, "An introduction to digital storage media—command and control," *IEEE Communications Magazine*, vol. 34, no. 11, pp. 122–127, 1996.

[21] R. J. Crinon, D. Bhat, D. Catapano, G. Thomas, J. T. Van Loo, and G. Bang, "Data broadcasting and interactive television," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 102–118, 2006.

[22] European Telecommunications Standards Institute, "DVB specification for data broadcasting," Tech. Rep. TS 101 192, ETSI, Cedex, France, June 1999.

[23] R. J. Crinon, "DSM-CC object carousel for broadcast data services," in *Proceedings of the 16th IEEE International Conference on Consumer Electronics (ICCE '97)*, pp. 246–247, Rosemont, Ill, USA, June 1997.

[24] H. Zhang, T. Jiang, Z. Gu, and S. Zheng, "Design and implementation of broadcast file system based on DSM-CC data carousel protocol," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 3, pp. 929–933, 2004.

[25] E. M. Schwalb, *iTV Handbook: Technologies and Standards*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2003.

[26] Sun Microsystems, "Jar File Specification," 1999.

[27] PKWare, "Zip File Format Specification Version 6.3.0," 2006.

[28] P. Newton, D. Kelly, J. Tan, J. Shi, and L. Gan, "Recording interactive TV," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 2, pp. 540–544, 2005.

[29] J. Tan, J. Shi, L. Gan, D. Kell, and P. Newton, "Solutions and systems for recording interactive TV," in *Procedings of the 7th International Symposium on Optical Storage (ISOS '05)*, vol. 5966 of *Proceedings of SPIE*, pp. 1–7, Zhanjiang, China, April 2005.

[30] D.-H. Park, T.-Y. Ku, and K.-D. Moon, "Real-time carousel caching and monitoring in data broadcasting," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 1, pp. 144–149, 2006.

[31] Z. Mai, P. Nasiopoulos, R. K. Ward, and S. Infante, "Real-time DVB-MHP to blu-ray system information transcoding," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, pp. 639–647, 2008.

[32] Cineca Interuniversity Consortium, "Just DVB-It," May 2007, http://www.cineca.tv/labs/mhplab/JustDVb-It 2.0.html.