A Resource Identification Mechanism for Interactive DTV Systems

Marcio Ferreira Moreno, Rogério Ferreira Rodrigues, Luiz Fernando Gomes Soares
Departamento de Informática – PUC-Rio
P.O. Box 38.097 – 22.453-900 – Rio de Janeiro – RJ – Brasil
{mfmoreno,rogerio,lfgs}@inf.puc-rio.br

A Resource Identification Mechanism for Interactive DTV Systems

Marcio Ferreira Moreno, Rogério Ferreira Rodrigues, Luiz Fernando Gomes Soares
Departamento de Informática – PUC-Rio
P.O. Box 38.097 – 22.453-900 – Rio de Janeiro – RJ – Brasil
{mfmoreno,rogerio,lfgs}@inf.puc-rio.br

Abstract

This paper describes a resource identification mechanism for interactive applications running on broadcast digital TV systems. The proposed mechanism relieves application authors from having knowledge of the addressing scheme used to identify resources in DTV presentation environments (set-top boxes or other receivers) and transport schemes based on MPEG-2 Systems and DSM-CC standards. The proposal comes from the reference middleware defined for the Terrestrial DTV Brazilian System, but can be applied to other DTV systems.

1. Introduction

Most of terrestrial digital TV systems use MPEG-2 Systems [1] to multiplex main audio, main video and data elementary streams into a flow named transport stream (MPEG-2 TS), which is then modulated and transmitted. Data streams that are not related with the audiovisual main streams via timestamps can be sent cyclically within a TS flow, in order to allow its reception independent from the TV channel tuning time. MPEG-2 DSM-CC protocol (Digital Storage Media – Command and Control) [2] supports this cyclical data transmission through a protocol named data carousel.

An application together with its referred resources can be represented by a unique file system, or by more than one file system having different parent directory. One of the main characteristics of DSM-CC is to allow a cyclical transmission of a file system through an abstraction known as object carousel.

More than one object carousel can be transmitted simultaneously, and each one can refer to resources (files, directories, and other objects [2]) carried in other object carousels. Moreover, applications in a carousel can refer to resources in other carousels. For example, an HTML page transmitted in an object carousel can refer to an image whose content is transported in another object carousel [3].

In order to support these inter carousel references, it is necessary to translate the resource identifier used in the authoring platform to the one used in the transmission structure, and afterwards, from this last structure to the identifier used in the presentation (client) environment.

In order to run an application, a TV receiver must tune the desired frequency channel, decode the received object carousel data stream, extract the corresponding file system and place it in a memory location from where the application can be triggered and its data referred. The file system structure must be preserved in all this process, and more than that, it should support the same reference arrangement created in the application authoring phase.

It should be stressed that, although object carousels maintain the same file and directory structures of the content producer platform, the file system root is lost, when the carousel is generated. Without additional information, a receiver is not able to know under which parent directory the received carousel file system root should be placed, in order to maintain the same references specified by the application author.

If an application and all its referred resources are transported in a unique object carousel, and if all references made by the author are specified using relative URLs (Universal Resource Locators) [4], the references made by the application author need not to be changed, and the application file (or files) can be transported in the carousel as it was generated during the authoring phase.

However, its is usual to refer resources using non-relative URLs, and sometimes this is the unique alternative, as for example, when referred resources are placed in content providers different from the authoring platform. When file systems that contains these resources are transported via object carousels, their root parent are lost and their corresponding URLs specified by the author lose their meaning.

One solution is translating the application's URLs to locators used by object carousels that transport the resources, altering the original application specification. In the current terrestrial DTV systems, translations are done in the authoring (server) side, automatically or requiring author interventions. This strategy overloads the authoring environment, may require the author knowledge about the transport system, may cause specification errors and may add undesirable delays in the generation and transmission of DTV applications.

This paper proposes an alternative mechanism to solve, or reduce, the aforementioned problems. The proposal is part of the data coding specification of the Brazilian variant of ISDB (International System for Digital Broadcasting), which has guided the reference implementation of the Brazilian DTV middleware, named Ginga.

The paper is organized as follows. Section 2 discusses related works. Section 3 presents the proposed solution and discusses its implementation. Section 4 is reserved for final remarks.

2. Related Works

There are two approaches defined by all the three main DTV Systems — the DVB system (Digital Video Broadcast), ATSC (Advanced Television System Committee) and ARIB (Association of Radio Industries and Businesses) — for translating author resource identifications into the identification scheme used by the presentation environment.

The first approach translates resource URLs defined in the authoring phase into proper Locator URLs (each system has a different scheme to refer to a resource in an object carousel [5, 6, 7, 8]), which identifies the same resources in object carousels. The translation may trust on the author knowledge of how to identify application resources using the DSM-CC protocol object carousel. However, user intervention is an error-prone procedure, since the resource location syntax is based on hexadecimal codes (for example, in the DVB Locator: original_network_id, transport_stream_id, service_id, etc [5]), and requires the author knowledge about the transport system. Alternatively, an authoring tool may automatically translate application URLs into Locator URLs. In both cases, the original document will contain URL strings that do not make sense for the authoring environment file system. Moreover, as URLs are tied to the DSM-CC carousel, it makes difficult to test applications in the broadcaster (content producer) site. It is important to note that if the generated application needs to be stored in the client receiver for a future exhibition, after translating the URLs, a new file system must be created (in the receiver storage device), which has no relation with the original file system, since the translation into object carousel URLs breaks the relationship with the original locators. Thus, future editions of this content by the server author would be very hard to be done, if not impossible.

A second approach is based on *location descriptor* mechanisms, also defined in the three DTV system standards, as described in the next five paragraphs.

In order to avoid application authors knowing the exact representation form of the involved identifiers in the *DVB Locator*, the European standard specifies a *location descriptor* mechanism that allows a simpler description both for procedural interactive applications (*dvb_j_application_location_descriptor*), and for declarative interactive applications (*dvb_html_application_location_descriptor*) [5]. However, this mechanism is restricted to resources under the same and unique root.

The *dvb_j_application_location_descriptor* has three core fields: *base_directory*, *classpath_extension* and *initial_class*. The first one contains the directory name, relative to the root "/" directory, which will be the base directory for relative paths. This directory is automatically inserted into the *classpath* variable for Java interactive applications. The second field (*classpath_extension*) is optional and contains a string specifying a directory name relative to the base directory, which should also be added to the Java application *classpath*. The *initial_class* field carries the name of the Java class that starts the application.

The *dvb_html_application_location_descriptor* core fields are: *physical_root* and *initial_path*. The *physical_root* contains a string defining the application root directory. Analogous to the procedural application descriptor, if the application base directory is the root directory, the field should contain the string "/". The *initial_path* field contains the path, relative to the *physical_root* path, for the declarative document that initiates the declarative application. For example, consider a transmission that has the "/application" directory, containing two subdirectories: "images/" and "main/". The "main/" subdirectory contains the "index.htm" document and the "images/" subdirectory contains figures composing the declarative application. In this case, the *physical_root* field has the "/application/" value while the *initial_path* has the "main/index.html" value.

The ACAP and ARIB standards also define the same *location descriptor* (syntax and semantics) used in the European standard for locating applications. Nevertheless, the Japanese system does not specify any kind of descriptor for declarative applications, requiring the use of the complete Locator [7], as discussed in the first approach.

Location descriptors can be generated automatically or with an author intervention. They can be then multiplexed into the transport stream, allowing to maintaining in the client-side the same references used in the server-side, since now an extra information is sent to the client side in order to recover the root path of all references As an example, the object carousel generator of Tektronix [9] is based on this strategy. In this tool, the author fills a form specifying the values for the application initial path and the physical root directory. From these field values, the carousel generator creates an object to be sent in the broadcast transmission containing the translation of the relative URLs into carousel URLs. In the client-side, the relative URLs must be translated into the carousel URLs, before starting the application or on the fly.

This second approach has, however, an important drawback. When defining the base directory, it becomes impossible to use absolute references for application objects that do not belong to the file tree structure having the base directory as root. Thus, without modification in the resource identifiers, the mechanism does not support the transmission of applications that refer to content stored in network shared devices; or resources stored in different hard disks or partitions of a same machine, even if these resources are being transmitted in a different object carousel.

In order to improve this second approach, the authoring tool should perform one of two alternatives. The first one is to create another file system grouping all files referred by an application. In this case it would be impossible to reuse a file sent in another object carousel. Of course, a delay would be introduced and a data structure would be necessary to store this new created file system. The second alternative is to create a new version of the application, in the authoring environment, with different identifiers that only would make sense in the presentation (client) environment. This new version would be created translating absolute URLs in the server side in relative URLs for the client side using the location descriptor definition. Again, this approach overloads the authoring environment and depends that authors fill descriptors information. Hence, as in the first alternative, this may introduce a delay in the carousel generation process that can make impossible to maintain temporal relationships in interactive applications, in particular for live content generation and transmission.

Different from the two aforementioned approaches, the Brazilian DTV system relies on a new mechanism that does not overload the authoring tool and neither the carousel generator, and, at the same time, does not oblige application authors to know details of how to identify and locate resources in presentation environments and transport systems. The proposed mechanism is detailed in the next section.

3. The Proposed Mechanism

The mechanism proposed in this paper will be explained based on the declarative environment reference implementation of Ginga – the Brazilian DTV middleware standard – and the implementation of a datacasting server for this DTV system. Only the relevant broadcast transmission functions of Ginga related with this proposal will be presented. Data transmissions through the return channel will be omitted.

In the declarative environment of Ginga, an application is written using the NCL language. NCL (Nested Context Language) [10] is an XML based glue language used to define spatial and temporal synchronization among media objects (video, audio, image, text and procedural code objects), and to define alternative content and alternative presentations for application media objects.

When an interactive application needs to be transmitted via object carousel, an event object [2] must be created reporting which stream event descriptors [2] will be associated with the application. In Ginga, an event of type “gingaEditingCommand” is used. The receiver middleware must register itself as a listener of this type of event.

The event object must be created mapping the “gingaEditingCommand” string into a DSM-CC stream event descriptor identifier chosen by the carousel generator. The event object must be transmitted in the same object carousel that carries the initial interactive application file.

One or more stream event descriptors, with the same identifier defined in the event object, are then transmitted carrying editing commands in its payload field [2]. These commands will be responsible for supporting the mapping between the data structures used in the server side (broadcaster environment) to the client side data locations (receiver environment). Thus, they are the triggers of the mechanism proposed in this paper, as explained in what follows.

In Ginga, two particular editing commands are used to load media and application contents: the “addDocument” command, used to load an NCL application and some of its content resources; and the “addNode” command, used to add a new data structure to a previously loaded NCL application. This new data structure can refer to resources (media contents) that need to be handled by the application (the NCL document). The new data structures as well as the interactive application are XML (eXtensible Markup Language) documents [11].

A set of editing command parameters is used to bind the XML document (an interactive application or a new data structure to be added to a previously loaded application) location, in the server platform, to its location in the object carousel used for the data structure transmission. These command parameters are <URL, IOR> pairs, where the URL has the DTV scheme ("x-sbtvd" in the Ginga example), followed by the absolute path of the XML document to be transmitted, and the IOR refers to the specific object in the DSM-CC object carousel (service domain id, module id and object id [2]) that carries the resource. Ginga can also obtain the XML document via the return channel, without using DSM-CC object carousel, but this is unimportant regarding this paper proposal.

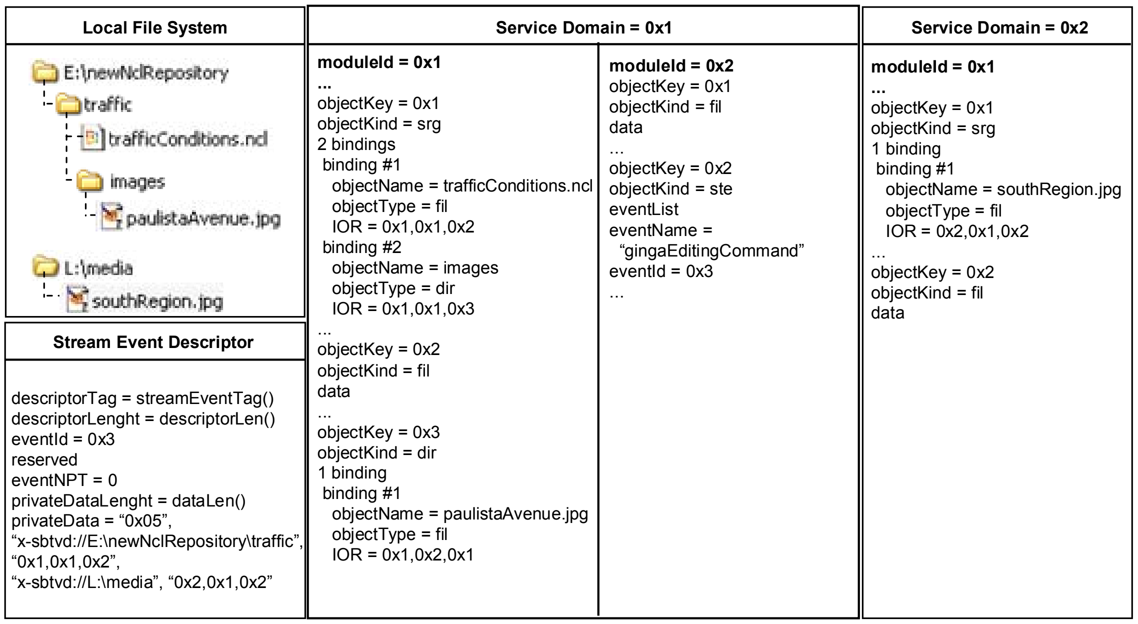
To deal with all the identifier references used by applications (case of the "addDocument" command) and by the extra added XML data structure (case of the "addNode" command), additional file systems may have to be transmitted in different object carousels. In this situation, other <URL, IOR> tuples must be added into the parameter set of the editing command. The URL should have the "x-sbtvd" scheme, followed by the absolute path (as seen in the authoring environment) of the root of the file system. The corresponding IOR should refer to the object (in the object carousel) that represents the resource content. Indeed, these new <URL, IOR> pairs are necessary when the file system organization that represents the application and all of its content cannot be suitably represented by a single tree, as in the example presented in Figure 1. In the example, the "trafficConditions.ncl" application refers to the "southRegion.jpg" resource. However, the two files need to be transmitted in different carousels [2].

All this procedure is automatically performed by the datacasting server, without any author intervention. The author can thus concentrate only on the application conception and the specification of its file systems to be transmitted.

Upon receiving the editing command, the receiver middleware parses the set of <URL,IOR> pairs and creates a table relating each received URL with the new memory location URL where the object referred by the IOR was, or will be, mounted. From this moment on, when the received application executes, all URLs it refers will be resolved, using the created table, into new URLs corresponding to the client storage devices. It is worth mentioning that the application document remains as it was in the server side, that is, its URLs both in the authoring and the presentation environment have the same values. The translation (mapping) is realized automatically by the middleware. It is also important to note that, different from the other digital TV system solutions, any knowledge about the object carousel complex structures is not required from an author, and that it is not necessary to generate a new version of the interactive application before its transmission, with resource identifiers that make sense only in the receiver side. The proposal is simple, but brings an important change in the paradigm used by the current digital TV systems: resource location resolution is not anymore done in server-side, but in client-side.

Figure 1 describes an example of interactive application transmission that uses the proposed mechanism. In the example, the content provider wishes to send an interactive application ("trafficConditions.ncl") stored in one of the broadcaster datacasting servers (in a local file system), but referring resources in different hard disks or partitions, as illustrated in Figure. Two object carousels are generated to carry the interactive content (Service Domain = 0x1 and Service Domain = 0x2). Besides the application and part of their content, the object carousel represented by Service Domain = 1 carries an event object. This object relates the event type "gingaEditingCommand" with the event identifier "0x3".

The stream event descriptor (Figure 1) is transmitted with the identifier "0x3", as specified by the event object of carousel "0x1". In this event, the value "0x0" is assigned to the temporal reference (*eventNPT* field in Figure 1), requiring its immediate execution. The private data field carries the "addDocument" command code and a set of <URL, IOR> pairs. In the first pair, the URL has the "x-sbtvd" scheme and the local absolute path "e:\newNclRepository\traffic" related with the interactive application. The associated IOR locates the object that carries the application: Service Domain "0x1", module "0x1" and object "0x2". In the second pair, the URL has the "x-sbtvd" scheme and the "L:\media" path, while the associated IOR locates the image "southRegion.jpg" in Service Domain "0x2", module "0x1" and object "0x2". Using this information, the middleware stores the received absolute path of each received URL, relating them with the local memory area URL where the objects, passed on each transmitted IOR, was (or will be) mounted.


```
Local File System
E:\newNclRepository
  traffic
    trafficConditions.ncl
    images
      paulistaAvenue.jpg
L:\media
  southRegion.jpg
```

```
Stream Event Descriptor
descriptorTag = streamEventTag()
descriptorLenght = descriptorLen()
eventId = 0x3
reserved
eventNPT = 0
privateDataLenght = dataLen()
privateData = "0x05",
"x-sbtvd://E:\newNclRepository\traffic",
"0x1,0x1,0x2",
"x-sbtvd://L:\media", "0x2,0x1,0x2"
```

```
Service Domain = 0x1
moduleId = 0x1
...
objectKey = 0x1
objectKind = srg
2 bindings
 binding #1
   objectName = trafficConditions.ncl
   objectType = fil
   IOR = 0x1,0x1,0x2
 binding #2
   objectName = images
   objectType = dir
   IOR = 0x1,0x1,0x3
...
objectKey = 0x2
objectKind = fil
data
...
objectKey = 0x3
objectKind = dir
1 binding
 binding #1
   objectName = paulistaAvenue.jpg
   objectType = fil
   IOR = 0x1,0x2,0x1

moduleId = 0x2
objectKey = 0x1
objectKind = fil
data
...
objectKey = 0x2
objectKind = ste
eventList
eventName =
  "gingaEditingCommand"
eventId = 0x3
...
```

```
Service Domain = 0x2
moduleId = 0x1
...
objectKey = 0x1
objectKind = srg
1 binding
 binding #1
   objectName = southRegion.jpg
   objectType = fil
   IOR = 0x2,0x1,0x2
...
objectKey = 0x2
objectKind = fil
data
```

Figure 1. Example of the proposed mechanism

Different from other digital TV system mechanisms, the resource identification is performed in the receiver exactly as it is performed in the authoring environment, in spite of if the interactive application works with absolute or relative URLs, or if the object carousel which is being used refers to resources in other carousels. For example, the "trafficConditions.ncl" application could refer to the "brazilianMap.jpg" relatively or absolutely. Furthermore, the application could refer to a content stored in a different provider, which would be loaded through a different object carousel. The resource location mapping is automatically performed by the middleware.

Finally, it is important to mention that this paper proposal is completely compatible with the other aforementioned digital TV systems. Nevertheless, small changes in their implementations are necessary, since a component to maintain the data structure that relates the server-side and client-side location strategies is required.

4. Final Remarks

DSM-CC object carousel is the protocol commonly used for broadcasting interactive applications in digital TV systems. When received by the client systems (set-top boxes, cell phones, etc.), this applications are put in the client system memory/storage device, i.e., in locations different from the ones referred in the original applications.

Although allowing that applications correctly refer to resources in the presentation environment, the main digital TV standards define complex schemes to identify these resources using absolute paths that require a complete redefinition of the locators used in the authoring environment. As an alternative, mechanisms for identifying resources using relative paths have been defined by all main DTV systems. However, such mechanisms may introduce delays in the object carousel generation and may overload the authoring environment, due to the possible requirement for creating new versions of the application file structures, for creating descriptors, for PSIs (Program Service Information) table [1] handling and for multiplexing all these structures into the transport stream.

In contrast, the proposal of this paper consists on a mechanism that allows DTV interactive applications to refer their resources identically, in spite of they are in the authoring environment or in the presentation environment. To accomplish that, the resource reference mapping process is done in the client side, instead of being realized in the server side as occur in the other DTV system proposals. The datacasting server uses features of the DSM-CC protocol to transmit the necessary information for using unique locators. In the receivers, the middleware maintains a data structure capable of translating the transmitted identification in locators that respect the receiver environment characteristics. The proposed mechanism has been validated through the reference implementation of the Brazilian Digital TV middleware, named Ginga, and through the reference implementation of a datacasting server for the same DTV system.

Among the main contributions of this paper we can highlight the high-level abstraction offered to authors, which become free of being concerned with issues related to resource identification in different environments. Furthermore, the proposed mechanism allows interactive applications to be broadcasted without the need of changing their resource identifiers, even if the resources are located in network distributed storage systems. As a benefit, the delay introduced when generating the transport stream and/or when creating new versions of applications may be reduced.

As a future work we intend to implement the proposed mechanism for the Japanese DTV system middleware and, in this context, adapt datacasting servers for this system to have the parameters transmitted directly in private data field of DSM-CC event descriptors. This would be a contribution in the direction of defining an outgoing international standard (ISDB) based on a joint Japanese-Brazilian digital TV effort.

5. References

[1] ISO/IEC 13818-1. Information technology – Generic coding of moving pictures and associated audio information - Part 1: Systems, 2000.

[2] ISO/IEC 13818-6. Information technology – Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC, 1998.

[3] Interactive TV Standards. Focal Press, Elsevier. Morris, S. and Chaigneau, A. S, 2005.

[4] Request for Comments: 3986. Uniform Resource Identifier (URI): Generic Syntax, 2005.

[5] Digital Video Broadcasting (DVB), Multimedia Home Platform (MHP) Specification 1.1.1, ETSI TS 102 812, ETSI Standard, 2003.

[6] Advanced Application Platform (ACAP), Document A/101, ATSC Standard, 2005.

[7] ARIB STD-B24, Version 3.2, Volume 3: Data Coding and Transmission Specification for Digital Broadcasting, ARIB Standard, 2002.

[8] ARIB STD-B23, Application Execution Engine Platform for Digital Broadcasting, ARIB Standard, 2004.

[9] AD951A/AD953A MPEG System User Manual – Carousel Generator, Tektronix, 2000.

[10] ISDTV-T Data Codification and Transmission Specifications for Digital Broadcasting, Volume 2 – GINGA-NCL XML Application Language for Application Coding. Sao Paulo, SP, Brazil. Soares, L.F.G., 2006.

[11] Live Editing of Hypermedia Documents. Proceedings of ACM Symposium on Document Engineering, p. 165-172. Costa et al, 2006.