J Internet Serv Appl (2010) 1: 69–79
DOI 10.1007/s13174-010-0002-y

ORIGINAL PAPER

Towards the convergence of digital TV systems

Luiz Fernando Gomes Soares · Marcelo Ferreira Moreno · Romualdo Monteiro de Resende Costa · Marcio Ferreira Moreno

Received: 20 December 2009 / Accepted: 31 January 2010 / Published online: 20 April 2010

Abstract To allow producing digital TV applications independently from receiver's hardware and operating system, and also to provide better support to application designs, middleware layer is introduced in digital TV system architectures. At first, middleware systems were developed aiming at specific transport platforms (IPTV, terrestrial DTV, etc.), offering support to services specifically designed for those platforms. However, the next generation of digital TV pulls all TV services present in all current platforms together into a single core of distributed services, as a result of the transport platforms convergence. In this hybrid TV, transport systems shall be concealed by the middleware to applications, as other operating system and hardware resources are hidden.

This paper emphasizes the middleware natural role as key technology for this upcoming convergent digital TV, raising some requirements to be committed. NCL and Ginga-NCL features—technologies recommended by ITU-T for IPTV services, and ISDB standards for terrestrial DTV—are used as examples of some proposed solutions, as well as to illustrate some issues which deserve future research attention and new better results.

L.F.G. Soares (✉) · M.F. Moreno · R.M. de Resende Costa · M.F. Moreno
Depto. de Informática, PUC-Rio, Rua Marquês de São Vicente 225, 22453-900 Rio de Janeiro, RJ, Brazil
e-mail: lfgs@inf.puc-rio.br

M.F. Moreno
e-mail: moreno@telemidia.puc-rio.br

R.M. de Resende Costa
e-mail: romualdo@telemidia.puc-rio.br

M.F. Moreno
e-mail: marcio@telemidia.puc-rio.br

Keywords Convergent digital TV · Middleware · Declarative languages · Ginga-NCL

Abbreviations

DTV	Digital TV
NCL	Nested context language
LASeR	Lightweight application scene representation
SVG	Scalable vectors graphics
SMIL	Synchronized multimedia language
HTG	Hypermedia temporal graph
VoD	Video on demand
ESG	Electronic service guides
URL	Universal resource locators
DSM-CC	Digital storage media—command and control
ITU	International telecommunication union
ISDB	International standard for digital broadcasting
SBTVD	Brazilian DTV system

1 Introduction

Digital TV (DTV) systems have been reported in the literature under different names: IPTV, WebTV, Internet TV, Broadband TV, Terrestrial DTV, Satellite DTV, Broadcast TV, P2P TV, etc. These terms are employed depending basically on the transport platform used for pushed or pulled data transmissions; in other words, based on lower layers of a DTV reference model, as sketched in Fig. 1. However, constraints imposed by current transport infrastructures also make these systems different with regard to services they provide. For example, VoD (video on demand) services are the basis of IPTV systems, and are almost impossible to be provided in a pure terrestrial DTV or satellite DTV.

Usually, the transport platform and the decoding of high-quality main audiovisual streams are implemented in hard-

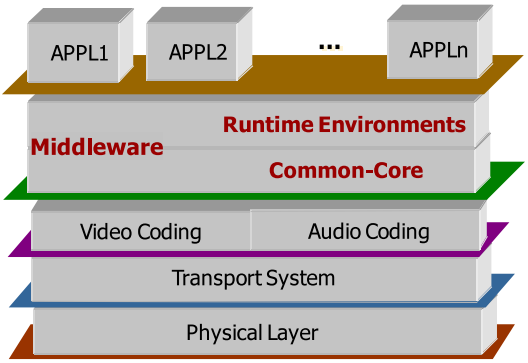

Fig. 1 DTV reference model

ware. To allow DTV applications to be developed independently from the receiver hardware and operating system, and also to provide better support to application design, a *middleware* layer is introduced in the DTV reference model shown in Fig. 1.

At first, middleware systems were developed aiming at specific transport platforms, offering support for services/applications specifically designed for those platforms. However, convergence is also inexorable in the DTV domain. Hybrid broadband/broadcast systems begin to appear, although they still comprise diffident solutions.

The next generation of DTV systems will put together all TV services present in all current platforms, as a consequence of the transport platform convergence. The transport platform shall be concealed by the common-core middleware sublayer to the runtime environment middleware sublayer (the API offered to applications) and, as a consequence, hidden to applications; likewise other operating systems and hardware resources are concealed. Of course, at content producer side, applications should still be designed aiming at specific domains of receivers, taking into account their resource limitations, including their networking capabilities.

Probably we will continue to have names like IPTV, WebTV, etc., but only stressing business models and not underlying technologies. From the technological point of view we will have DTV systems, without adjectives, embracing all current solutions and services, and enhancing them. This paper focuses on the middleware designed for these broader systems.

Some efforts in this direction have already started in ITU-T, as reported in its H.760 series (Study Group 16—Multimedia Coding, Systems and Applications) [1]. H.760 series addresses a framework, for service development; a glue language called NCL (Nested Context Language) [2], as the integration mechanism; and the language engine, called Ginga-NCL [3].

We intend in this paper to raise some middleware requirements for the next generation of DTV systems. More specifically, we discuss some issues still open and future trends, considering the middleware support to: (i) applications designed for multiple networked exhibition devices; (ii) context-aware applications; (ii) 3D environments; (iv) fine-grained and temporal-consistent presentation control; and (v) data processing, both for pushed and pulled data, including intermedia QoS requirements.

In the remainder of the paper, NCL and Ginga-NCL features are used as examples of recent solutions proposed by us, as well as to illustrate some issues that deserve future research attention and new better solutions. After a brief introduction to DTV applications in Sect. 2, this paper follows the top-down description of the middleware sublayers depicted in Fig. 1. Application design methods and usability analysis are out of the scope of this paper. Section 3 discusses some issues regarding the support offered by the middleware runtime environments to applications. Section 4 deals with the middleware common-core and some embedding problems to be solved. Section 5 briefly addresses the transport platform support, in what it may have an influence on the middleware intermedia temporal consistency. Physical layer subjects are also outside the scope of this paper. Section 6 presents our final conclusions.

2 DTV applications: an overview

Applications developed for DTV may be associated with a particular TV service (or channel). In this case they are usually called *bound applications*. When a viewer changes to another service (channel), applications bound to the previous service are deactivated or finished. In contrast, other DTV applications are not associated with a particular service. They are present across all of the TV services (channels). In this case, they are called *unbound applications*. As an example, widgets may be loaded anytime during any TV service exhibition.

Bound DTV applications may be completely independent from the main audiovisual stream being transmitted: both semantically and temporally. As for example, an advertisement about nighttime TV program schedule may be presented at anytime during the morning TV programs of the same channel. Indeed, ESGs (Electronic Service Guides) [4] may compose a particular bound application for a specific TV channel, but they may also be examples of unbound applications, when they refer to several different channels (services).

On the other hand, bound applications may be temporally independent but semantically dependent of the main audiovisual stream being transmitted like, for example, when supplementary news about a TV program may be accessed during the program exhibition.

Still other bound applications may be both temporally and semantically dependent of the main audiovisual infor-

mation in exhibition, like when a product merchandize appears at the moment the product is being exhibited (or mentioned) in the main audiovisual stream.

In all mentioned application types, viewer interactions can be allowed, and in all cases media assets (video, audio, images, text, etc.) that compose an application can be related in time and space, no matter they are related or not to the main audiovisual stream.

Bound applications usually need middleware support. Unbound applications may be native, or received as pulled or pushed data. Native applications are usually written for the operating system of a receiver rather than the middleware, but they may also be supported by the middleware. Non-native unbound applications usually need middleware support.

DTV applications on the whole can be partitioned into a set of declarative applications and a set of imperative applications. Declarative applications are those whose initial entities are of declarative content type. Imperative applications are those whose initial entities are of imperative content type.

Declarative languages emphasize the high-level description of an application rather than its decomposition into an algorithmic implementation. Moreover, declarative languages usually define specific models to design applications targeted at specific domains (a declarative DSL—Domain Specific Language) offering a good balance between flexibility and simplicity. In other words, one loses some expressiveness but gains simplicity. For particular cases not covered by the declarative domain, declarative languages usually include embedded scripting language support.

Imperative languages like Java, which rely on a virtual machine to achieve portability, have engines that are very resource consuming, and they usually require considerable memory footprint. Both requirements can be a significant problem for low-end receivers.

Authoring DTV applications using imperative languages is more complex and more error-prone than when using declarative domain specific languages. Declarative descriptions are easier to be devised and understood than imperative ones, which usually require intended programming expertise.

Content and DTV application producers are usually inexperienced programmers. In addition, in Social TV applications [5–7], viewers can become producers or co-producers. Therefore, besides having a good and lightweight graphic authoring tool, the authoring language knowledge is essential for producing attractive applications. Declarative languages are valuable in this perspective.

Several declarative middleware solutions rely on XHTML. However, XHTML carries a legacy from previous technologies developed for text navigation and formatting, and has lots of add-ons created to overcome its limitations in the DTV domain. XHTML is focused on user-interaction declarative support as a means of synchronizing media assets' presentations. This narrow declarative scope forces application authors to solve spatiotemporal synchronization that goes beyond simple user interactions, as well as to solve content and presentation adaptations, and other issues usually found in DTV applications, by using imperative objects, usually written in ECMAScript. Thus, the great advantage of using a declarative DSL is lost, with the additional expense of using a scripting language with high CPU and memory requirements.

A declarative DSL approach that fulfills the main requirements of DTV applications, relegating for the imperative approach only particular computations, seems to be the right solution for a DTV middleware API. This approach would boost integration, simplicity and better resource usage in DTV platforms. Besides that, it would make the authoring process easier and less error-prone.

The Nested Context Language (NCL) [8] and MPEG-4 LASeR (Lightweight Application Scene Representation) [9] are technologies currently closest to fulfill these requirements. Based on SVG (Scalable Vector Graphics) [9] and other extensions [9], LASeR has its focus on media synchronization, as well as NCL. Both languages support content and presentation adaptability, and provide support for live editing commands [2]. NCL reuse facilities [10] are more versatile than the LASeR ones. NCL also offers a more powerful support for applications targeting multiple exhibition devices. Despite being a good solution, mainly for mobile devices, LASeR does not have a commercial implementation yet.

NCL is a *declarative* XML-based language initially designed aiming at hypermedia document specification for the Web. In 2007, NCL was adopted in the Brazilian terrestrial DTV standard, SBTVD [11]. In the beginning of 2009, NCL and its user agent, called Ginga-NCL, became part of ISDB standards (the previously known Japanese DTV standard now increased with Brazilian improvements) and part of ITU-R BT 1699 Recommendation [12]. Also in 2009, NCL and Ginga-NCL became the first standardized technology of the ITU-T multimedia application framework for IPTV services [1], in its ITU-T Recommendation H.761 [8]. NCL and Ginga-NCL have been designed at the TeleMidia Lab at PUC-Rio. The work has been coordinated by the authors of this paper who also chaired the ITU-T Recommendation H.761 and the Brazilian DTV Middleware Working Group. Ginga-NCL and NCL specifications are open and totally royalty-free [13].

The NCL flexibility, its reuse facility, multi-device support, application content and presentation adaptability, and mainly its intrinsic ability for easily defining spatiotemporal synchronization among media assets, including viewer interactions, make it an outstanding solution for all kinds of

DTV systems. In addition, NCL provides an API that allows for building and modifying applications on-the-fly through live editing commands. For particular procedural needs, as for example when dynamic content generation is required, NCL provides the Lua scripting language [14] support.

In the remainder of this paper, our proposals for NCL and Ginga-NCL are together used as examples to raise some open issues and to delineate some solutions. They are also used to introduce other solutions proposed in the literature.

As a glue language, NCL does not restrict or prescribe any media-object content type. Thus NCL may integrate (embed) code chunks (and solutions) written in any other programming language. NCL applications just define how media objects are structured and related, in time and space. In this sense, perceptual objects (image, video, audio and text objects), imperative objects (Xlet, Lua, etc.), and declarative objects (XHTML, SMIL, SVG, X3D, etc.) are supported by the language. Which objects are supported depends only on which object players (engines) are coupled to the NCL formatter (player). One of these objects is the one containing Lua code. As mentioned above, Lua [14] is the efficient and lightweight scripting language of NCL, used when algorithmic computations are needed.

3 Middleware runtime environments

Much work remains to be done regarding middleware support offered to applications. Some of them related to middleware runtime environments are discussed in what follows.

3.1 Multiple exhibition devices

The multiple exhibition device support provided by some languages [2, 15, 16] to allow home-area networking exhibitions (and beyond home-area distributed exhibitions) still deserves much attention.

The SMIL 3.0 [16] MultiWindowLayout module allows for defining elements and attributes to describe multiple sets of regions, in which multimedia content is presented. This approach is similar to the NCL [17] solution. However, SMIL does not allow authors to specify the association between a defined set of regions and a specific device or class of devices. The association can be done using a metalanguage interpreted by other engines, or without author interventions, using an algorithmic procedure (based on environment features, viewer preferences and presentation context [18]). The last case usually requires complex algorithms and is only possible for simple scenarios.

The LASeR [9] declarative specification follows the SVG [15] scene structure that can be fragmented in many access units, each one describing the time scene elements needed by the LASeR player. Like MPEG-4 BIFS [19], LASeR emphasizes the composition of media objects on one rendering device. They do not have a specific notation for multiple exhibition devices. However, considering that LASeR declarative specifications can be fragmented, fragmentation strategies [20] could be used to guide segment distribution over multiple devices.

The architecture *modus operandi* proposed for SMIL [16] has some similarities with the active class behavior proposed in NCL, in which the same initial content is presented in all devices, but individual and independent controls are allowed in each one. However, in the proposed architecture, SMIL application control is centralized by a player running in the server side. In Ginga-NCL the same approach can be adopted, but it also supports distributed control.

In addition to devices in active classes (that can be assigned to a centralized or distributed control), NCL also allows for registering devices in passive classes (in which the same content is presented in all its devices of a class under a single shared control). During application execution, the NCL formatter is responsible to find which devices are registered in which classes.

Thus, NCL goes one step ahead by defining classes of devices and allowing applications to distribute their content among these classes. The glue language characteristic of NCL allows for sending media objects (including those whose content is imperative or declarative code chunks) to specific classes of devices that are able to handle them. For example, we can have a distributed NCL application running part of its component on devices with SVG support, part on devices with X3D support, part on devices with Java Xlet support, part on devices with BML or any other XHTML-based language support, etc. A hierarchical model for device control is also defined for NCL [17]. Unlike the solution proposed by Cesar et al. [18], NCL does not need metalanguage support for content distribution.

However, in general, how devices register themselves in classes is an open issue, as well as how an application exposes which resources it demands from each class in order to advise device registration. Appropriate metadata ontology is necessary to allow for specifying application requirements concerning the use of multiple devices. This would let viewers (or the middleware, without human intervention) register secondary devices to receive each correct part of an application, instead of the strict and default solutions currently provided by some languages.

For example, the current Ginga-NCL reference implementation defines two classes by default; but it has no solution yet for the aforementioned problems closely related to metadata specification and processing. Better semantic descriptions of application contents are also needed for content and content presentation adaptations. They are also necessary for service-program guide generation and for conditional access control, as discussed in Sect. 3.

Cooperative editing is another open issue, in particular at the client side. To enable innovative social TV applications, it is essential to allow for secondary device communications and cooperative edition at the viewer side.

3.2 3D support

Little can be said about 3D support in DTV systems, other than 3D video rendering [21].

Languages like X3D [22] allow for building and interacting with 3D graphics. However, other 3D (and 2D) media objects must be related in time and space to DTV applications.

Although languages like NCL allow embedding 3D objects [23], middleware exhibition (graphic and video) plans are far from allowing a real 3D environment. Even the exhibition of 2D objects on 3D surfaces is only recently introduced and is still a novelty in DTV systems.

Temporal relationships between atomic 3D objects can be achieved in languages like SVG and NCL, but spatial relationships among all kinds of 3D objects still deserves a lot of research efforts.

3.3 Authoring language abstractions

Are we working with appropriate authoring language abstractions? An analysis of authoring language aspects as interface language for creating DTV applications also needs more research attention. A specification language is an artifact (a human-made object designed for a particular end) whose primary purpose is to represent and support information processing. Computer languages are artifacts that have a dual nature. They represent information in a referential sense, and they also construct information in a generative sense [24]. In practical terms, this dual nature leads us to include, in the analysis of computer languages, not only the linguistic constructs that it offers for specifying computer representations and processes, but also its operational semantics (i.e., what effect language constructs bring about when they are interpreted by a computer). A third aspect of computer languages that should be included in the analysis is the programming infrastructure that supports programmers in creating and interpreting language constructs. Although the latter can be certainly considered an external factor that is not really intrinsic to the language being analyzed (for we can always use different editors and CASE tools to produce programs, regardless of the language we are working with), we claim that program editors, for example, highlight and in an extensive way make explicit certain features of programming languages. This is especially true for declarative languages and their abstractions.

In Ref. [10] NCL usability is evaluated in the context of using three authoring environments: a non-specialized XML text editor, a specialized textual tool, and a graphical authoring tool. Instead of evaluating the NCL notation expressiveness per se, it is analyzed how the notation communicates (to users) its design principles and the intent of its designers with the support of different computer environments that provide the necessary infrastructure for NCL programming. However, the analysis is limited to the reuse features of NCL.

Among the existing methods to evaluate languages are the empirical (i.e., methods involving empirical observations of how people actually use the features provided by the language in real task situations, or at least realistic lab tasks) and the analytical ones (i.e., methods derived from theories, models or frameworks, in varying degrees of formality). A combination of methods is clearly the best choice to gain insight and understanding with respect to a language support. In [10] only analytical evaluation of NCL is done. The approach has been to apply analytical and empirical methods in sequence, to detect specific features of NCL that its designers were not aware of, and once detected, to empirically test these features with NCL users in different contexts of use. Only the first steps have been taken. Much work remains to be done.

3.4 Presentation control

Upon receiving a DTV application specification, with all spatial and temporal relationships among its objects defined, the language engine must try to guarantee the correct presentation. To support this task, several data structures are computed from the application specification. These data structures must represent all possible predictable and unpredictable events that occur during a presentation, such as the start of a media presentation, a viewer interaction, etc.

One of these data structures, called *presentation plan* in this paper, is responsible for supporting the presentation scheduling. During a DTV application presentation, all information gathered from viewers and from the receiver, all viewer answers, and all viewer interactions are collected, updating the computed presentation plan. Therefore, this data structure represents the current multimedia presentation state, which can be stored and later retrieved and resumed from the saved state. This is a common situation found in a DTV environment, as for example:

- When viewers are allowed to explicitly pause a DTV application and then resume it at some later time—possibly days or weeks later, and even on a different device, still preserving all actions previously done;
- In bound applications, when a viewer changes the TV service/channel, thus starting another application in the new service/channel, but then regrets and returns to the previous service/channel, resuming the application and inheriting all information previously given, all answers previ-

ously provided, all interactions previously done, all environment information previously set, etc.

DTV environments have some specific characteristics that must be taken into account when defining an efficient presentation data structure. In bound applications in which the main audiovisual stream is temporally related to the application start-up time, and this moment would have occurred before the TV program is tuned in, the application must start immediately, and as if it has been started in the correct moment in the past. Thus, the presentation data structure must allow an efficient (with a minimum possible delay) application starting, from any moment in time during its assigned period. In all other cases, applications must be started from their beginning.

Most DTV middleware implementations, in particular those using imperative languages, do not allow applications to initiate from a point other than their beginning. This is due to the fact that it is very difficult to compute the presentation plan in advance. The plan is built in parallel with the application presentation. Thus, when a service is selected and its associated application should have already started, the application is simply ignored.

A poor solution used in most DTV systems is to split bound DTV applications into several small ones fired along the time. This approach can only be used for simple applications and completely loses the application logical semantics. Moreover, the responsibility of splitting an application efficiently and generating the corresponding triggering commands is passed to application authors. This can be a very difficult and error-prone task. Note also that this approach does not solve the problem, since it is still necessary to start each small application from its beginning. The approach works as if the whole bound application had discrete times that are possible starting points, with the granularity of these possible starting points given by the whole application partitioning.

Declarative middleware should allow applications to start from whichever point in time, since it is possible to build presentation plans in advance. For example, in the Ginga-NCL reference implementation [13] a special data structure, a labeled digraph called HTG (Hypermedia Temporal Graph) is proposed [17] as the basis of all temporal data structures. From the HTG, user agents derive the presentation plan to orchestrate media content presentations that make up a DTV application. Besides allowing application to start from any internal point in time, HTG provides an efficient data structure that allows a document presentation to pause and then be resumed in a future time, considering all interactive actions and all alternative choices performed in the past. To the best of our knowledge, these features are provided only by the Ginga-NCL reference implementation. HTG is also used to derive other important control plans, as discussed in the next section.

4 Middleware common-core sublayer

The middleware common-core sublayer is responsible for hiding platform details from applications. To help the discussion presented in this section, Fig. 2 illustrates the components of the Ginga Common-Core.

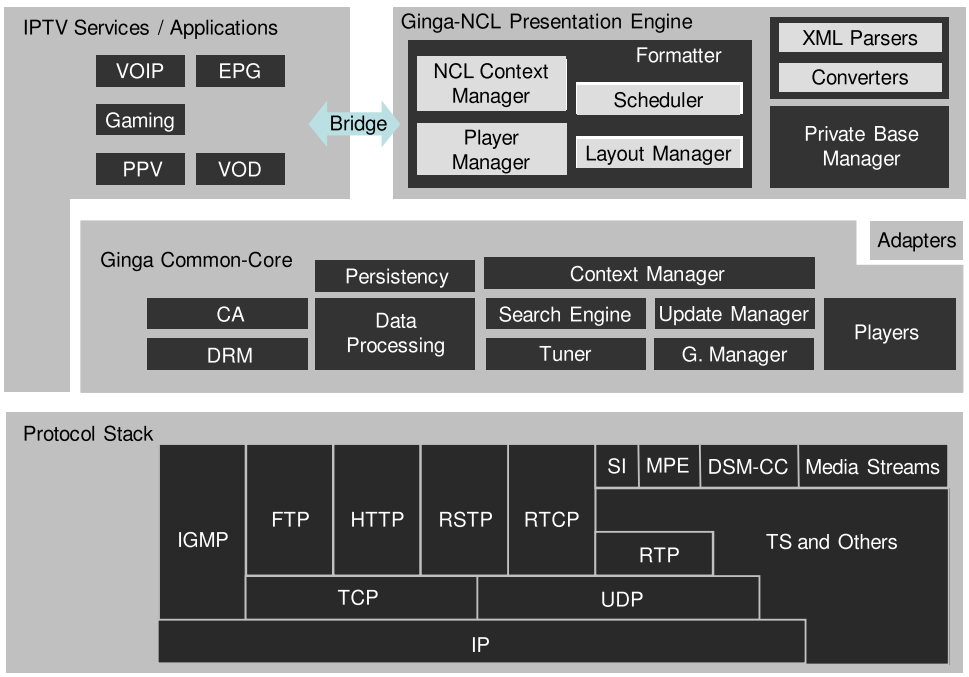

Fig. 2 Ginga architecture

4.1 Context management

The Context Manager component is responsible for generating information based on data gathered by its agents about platform characteristics, viewer profiles and available services. This information is used for content and content presentation adaptations, and for feeding other native applications, like VoD, ESG, recommender systems, etc.

In current DTV services there is still considerable dependency of limited structures provided by service information (SI) tables. (Service) Search engines are also biased by poor semantic descriptions of applications and services. Recommender systems are also biased by poor descriptions of viewer profiles. Conditional access mechanisms also suffer from the weak semantic descriptions of security and privacy polices.

Although content adaptations can be performed based on viewer profiles at the client side, it is still not possible to feedback these profiles to allow application authors to customize their services in server sides. Moreover, profile mining and exchanging in social TV environments is an open issue.

4.2 3D support

As mentioned in Sect. 2, good language support for 3D objects is still missing, in particular support to spatial relationship definitions among media objects. However, this lack of facilities is a consequence of poor Graphic Manager and Player tools defined by current DTV systems, which are responsible for managing all visual plans and for relating (overlaying) objects.

4.3 Data processing

Regarding Data Processing, there are several open issues deserving attention.

4.3.1 Resource identification

Pushed and pulled data must be received by the Data Processing module and placed in the local file system to be accessed by DTV applications. Usually, during the authoring phase, resources (video, image, text, audio and imperative code files) used to build DTV applications are located in the same platform where the authoring process takes place or in content providers that can be accessed by the authoring platform. URLs are commonly used to identify resources based on their locations. When these resources are received at the client (receiver) side, their non-relative URL identifiers usually have to be updated, since the resources will have a new local placement. The mapping between the original location (specified in the application specification) and the new location in the receiver is easy to be performed for pulled data (data received on demand), but it is not trivial for pushed data (unsolicited data reception).

In most DTV systems, pushed data that are not temporally related to the audiovisual stream via timestamps are sent cyclically in special streams. These streams, called object carousels in this paper, allow pushed data reception independently from the TV service selection time. For example, MPEG-2 DSM-CC protocol [25] supports this cyclical data transmission in all main terrestrial DTV systems.

Object carousels allow cyclical transmissions of file systems. A file system can store a DTV application specification file and all other content files referred by the application. A TV receiver that wants to run this application must be able to select the desired service, 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 accessed. The original file system structure must be preserved in this process to support the same reference arrangement created in the application-authoring phase. All these tasks are performed by the Data Processing module shown in Fig. 2.

Taking into account that object carousels and applications transmitted within these carousels can refer to resources in the same carousels or other carousels, it is also necessary to translate resource identifiers used in the authoring platform to those used in the transmission structure, and from these last ones to identifiers used in presentation (client) environments. Although object carousels usually maintain the same file and directory structures referred in the server platform, receivers are not able to know under which parent directory a file system root of a received carousel should be placed without extra metadata information.

Most terrestrial DTV systems provide a poor solution to this problem. Ginga-NCL provides a good solution [11] already tested for DSM-CC [25] carousels but it still needs to prove its efficiency for other pushed data protocols. IPTV systems until now have paid little attention to datacasting, since their main focus has been on VoD without embedded applications. However, this situation is changing in the new convergent DTV scenario. In VoD, data carousels should be transmitted to a multicast group other than the one used to transmit the audiovisual stream, since the cyclic characteristic of carousels allows them to be reused by several multicast groups with the same audiovisual content, but having a small time lag. Note that policies used to reduce network bandwidth in VoD services, like batch, patching, piggybacking, etc. [26], should be revisited for this new shared multicast channel.

4.3.2 Prefetching

Once application specifications are received, a middleware has two options: (i) to require all application contents be-

fore starting it; (ii) to require application media contents on-the-fly, that is, during application execution. The first solution requires a large receiver storage capacity, which is usually impossible for low-cost receivers. Moreover, such solution can introduce unbearable application-starting delay. The second solution is much better, but requires middleware to control content retrievals.

An intermediate poor solution, adopted in a large number of DTV systems, is the aforementioned split of a DTV application into several small applications fired along the time, with all limitations already mentioned in Sect. 2. This solution allows for storing the whole small-application content, one at a time, instead of the whole primary application content. The approach allows for using receivers with limited resources at the expense of delegating the receiver memory control to authors who must know how to split their applications to be played by receivers whose capacity they barely know. This can be a very difficult and error-prone task.

Thus, the better option is indeed to retrieve media contents during application runtime.

Concerning pulled data, download procedures depend on whether networks allow intramedia QoS negotiation or not. Intramedia QoS is discussed in the next section.

If QoS support is not provided for pulled data, receivers should download media objects guided by a *prefetching plan*. This plan is built based on the presentation plan discussed in Sect. 2, taking into account the estimated network transfer delay and jitter for each object. Since the plan is built based on estimations, prefetching in receivers' middleware is useful only to minimize the temporal mismatch probability between media object presentations. Of course, a conservative algorithm can avoid all temporal mismatches, but with a cost of more expensive receivers and larger application-starting delay. Indeed, bringing all application contents before starting the application can be considered a special case of this solution. When building the prefetching plan, a conservative approach should assume that all unpredictable events (like viewer interactions) happen immediately after they are enabled.

As prefetching plans are built based on estimations, there should be a monitor to compare the actual object prefetching duration with the expected duration previewed in the plan. If the actual duration overcomes the predicted one, the middleware should run elastic time algorithms [27, 28] to recalculate media object presentation durations, in order to maintain the temporal synchronization consistency.

For pushed data transported in carousels, prefetching plans establish when a media object should be taken out of a carousel. Carousel prefetching plans are built based on presentation plans, as usual, and on estimated carousel delays. Note that a carousel acts as a receiver's secondary memory, at the expense of wasting network bandwidth, as discussed in the next issue.

In the current Ginga-NCL reference implementation, a very simple procedure is used for object carousel prefetching, based on the worst carousel delay case. In this procedure, all unpredictable events are assumed to happen immediately after they are enabled. It must be stressed that like prefetching of pulled data, if prefetching plans for pushed data are built based on estimations, elastic time adjustments can also be necessary.

There are some proposals in the literature regarding prefetching, although few of them designed specifically for DTV and without any evaluation of their efficiency in this new domain. Furthermore, to the best of our knowledge, a prefetching algorithm that also takes into account elastic time adjustments is still an open issue.

4.3.3 Carousel management

Prefetching algorithms for pushed data assume that carousels transports all information needed to run an application. However, this is not a good approach, since it results in high bandwidth consumption and high access delay. In the case of carousels sent in broadcast channels, large carousels can leave a small bandwidth for main audiovisual streams, decreasing their quality. Therefore, it is worth trying to work with carousels containing only part of applications, the one that matches the current prefetching needs. This presumes that the server side knows which part of an application a carousel should transport at a certain moment.

Again, an intermediate poor solution for carousel management, adopted in a large number of DTV systems, is delegating the carousel management responsibility to application authors. Authors must split applications into smaller applications that in a whole give the same result. Each one of these smaller applications is then transmitted inside a carousel, as before. In this case, authors are also responsible for triggering these applications at precise moments. The carousel management is very simple in this case: small carousels for small split documents are created and transmitted. Note however that, besides worrying about prefetching problems, authors must also worry about carousel management when splitting applications; increasing even more the difficulty of this task.

An alternative and better solution is to run the carousel management autonomically. To accomplish this task, servers should build *carousel plans* to guide object insertions to or removals from carousels. Carousel plans would contain the moments in time when media objects should be available at receivers.

The Ginga server-side reference implementation adopts this solution, building its carousel plans based on the previously mentioned HTG model.

Carousel plans are similar to presentation plans built in receivers, with the exception that all unpredictable events

like viewer interactions must be treated as if they would happen at the moment they are enabled. This assures that all needed media will be in the carousel at the moment viewers interact.

From carousel plans, server-side middleware should estimate which objects must be placed inside carousels, how many times, in which places, and which objects must be removed; a difficult optimization problem indeed, without any solution reported in the literature, as far as we know.

The carousel bit length, the carousel stream transmission rate, and the space between same object instances give the maximum delay for retrieving the object. The carousel transmission rate is limited by the available network bandwidth. Moreover, if this bandwidth is shared with other data streams, as is the case when transmitting multiple carousels and data streams synchronized by timestamps (including main audiovisual streams) when the carousel transmission rate increases, the remaining bandwidth for other data streams decreases. As a consequence, the quality of service of these other streams can be put at risk, including the quality of the main audiovisual stream. Indeed, carousel removal and insertion is a big optimization problem that should also take into account other side effects. Moreover, carousel removal should take into account the probabilistic nature of the whole procedure to trigger appropriate elastic time media adjustments when it is necessary to maintain DTV application temporal consistency.

In the current Ginga server reference implementation, removal processes are carried out in their plenitude, but insertions certainly need better algorithms. In the current implementation, the object that carries the application specification is always present in the carousel. It is the only object inserted in several places, to minimize the application-starting delay. Other objects are inserted depending on their lengths; their maximal allowed retrieving delays; their expected presentation times, obtained from the carousel plan; and the carousel length.

It should be noted that using optimized carousels does not relieve receivers from managing downloads from these carousels. If they do not have sufficient memory to retrieve the whole carousels' data, prefetching plans should be built to guide retrievals, as previously discussed.

5 Transport-system protocol stack

As briefly mentioned in Sect. 3, download procedures for pulled data depend on whether the transport-system protocol stack allows intramedia QoS negotiation. Intramedia QoS deals with single media assets. It is associated with the moment contents are obtained from storage locations, network transfer rates, transfer delays and transfer jitters, in addition to scheduling policies defined by the involved (client and server) operating systems. In fact, intramedia QoS is an important feature to guarantee intermedia synchronization.

Unfortunately, bandwidth over-provisioning for main audiovisual streams and no guarantees for other media assets are the ways QoS is being treated in several current DTV systems. Maybe the key reason is that main audiovisual streams are considered the most important and the most demanding media content in current applications, and additional media objects still do not require strict QoS parameters.

Of course there are several studies and techniques to reduce bandwidth needs (almost always for main audiovisual streams), as those present in VoD services, like batching, patching, bridging, piggybacking, etc. Likewise, there are some suggested schemes for prefetching and carousel management. However, all these proposals do not take into account QoS negotiation procedures, although DTV systems have some interesting characteristics that could be explored towards a good solution.

In networks that offer QoS support, better control can be achieved. From presentation plans derived from application specifications, receivers can build their *QoS negotiation plans*, taking into account the transfer delay and jitter that will be negotiated for each media asset or set of media assets. QoS negotiation plans are used to trigger resource reservation procedures in order to obtain the desired QoS. If negotiation succeeds, it is guaranteed that media assets will be in a receiver on time; otherwise a new negotiation can be started with more relaxed QoS parameters or a new negotiation can be started in a future time, but with more strict QoS parameters.

Using resource reservation, the chances for temporal synchronization mismatches are reduced. However, they can happen and, in this case, elastic time adjustments can be needed if hard synchronization is demanded.

QoS negotiation in DTV systems for application content outside the main audiovisual stream brings back the interesting topic of resource reservations in advance as raised by mobile computing. Resource reservations in advance enable resource scheduling and allocation at an early stage in time. This way, resource availability can actually be guaranteed for the moment the resource is needed.

In mobile computing, handoffs can cause QoS breaks. Thus, resource reservations in advance should be based on future locations of mobile devices. The problem is to know future locations and when they will be reached. So, QoS in advance is based on estimations and with resource waste. In contrast, DTV systems do not have these constraints. Based on presentation plans, the exact moments in time for conducting resource reservations are known a priori, assuming that all unpredictable events happen immediately after they are enabled. Resource reservation in advance does not guarantee zero synchronization mismatches, but reduces the mis-

match probability, since it enlarges the time range for resource reservation negotiation.

6 Conclusion

The future of convergent DTV systems is near. However, there still is much work to be done to have an efficient and user-friendly system.

In these new DTV systems not only the main audiovisual stream will play an important role but also additional QoS demanding media objects, including those inserted by DTV application viewers, by using secondary networked devices. The next main drive will be to create an immersive environment in which social and personalized TV applications will be the core. In these applications, viewers will play a still more active role.

We tried to emphasize in this paper the natural role of the middleware as key technology for this upcoming convergent DTV in which services currently offered mainly in terrestrial DTV and IPTV systems will be present, enhanced and integrated into a single core of distributed services.

In addition, we present some research directions raising some issues which have no satisfactory solutions or no solution at all. These issues were categorized and discussed mostly in Sects. 3 and 4.

The raised points have been the focus of several research group efforts and standardization procedures. In particular they have been part of TeleMídia Lab efforts at PUC-Rio, the Ginga joint project at CTIC/MCT (Research Center for Information and Telecommunication Technologies of the Science and Technology Ministry of Brazil), and ITU-T H.760 series (Study Group 16) towards an interoperable convergent DTV framework.

References

1. ITU-T Recommendation H.760. Overview of Multimedia Application Frameworks for IPTV. Geneva, April 2009
2. Soares LFG, Rodrigues RF (2006) Nested context language 3.0, part 8: NCL digital TV profiles. Technical Report, Informatics Department of PUC-Rio, MCC 35/06, Rio de Janeiro, October 2006. http://www.ncl.org.br/documentos/NCL3.0-DTV.pdf
3. Soares LFG, Rodrigues RF, Moreno MF (2007) Ginga-NCL: the declarative environment of the Brazilian digital TV system. J Braz Comput Soc 12(4):37–46
4. Chiao H-T (2008) Comparison of the notification services between OMA BCAST 1.0 and DVB-IPDC phase 2. In: Proceedings of the 11th IEEE Singapore international conference on communication systems (ICCS '08), November 2008, pp 327–331
5. Mantzari E, Lekakos G, Vrechopoulos A, Social TV (2008) Introducing virtual socialization in the TV experience. In: Proceedings of the 1st international conference on designing interactive user experiences for TV and video (UXTV '08), Silicon Valley, CA, October 2008, vol 291. ACM, New York, pp 81–84
6. Geerts D, De Grooff D (2009) Supporting the social uses of television: sociability heuristics for social TV. In: Proceedings of the 27th international conference on human factors in computing systems (CHI '09), Boston, MA, April 2009. ACM, New York, pp 595–604
7. Harboe G, Massey N, Metcalf C, Wheatley D, Romano G (2008) The uses of social television. Comput Entertain 6(1):1–15
8. ITU-T Recommendation H.761. Nested context language (NCL) and Ginga-NCL for IPTV services. Geneva, April 2009
9. Dufourd J-C, Avaro O, Concolato C (2005) An MPEG standard for rich media services. IEEE Multimed J 12(4):60–68
10. Soares LFG, Soares Neto CS (2009) Nested context language 3.0—Reúso e importação. Technical Report, Informatics Department of PUC-Rio, MCC 33/09, Rio de Janeiro, March 2009. Also submitted to the Journal of the Brazilian Computing Society, as "Analyzing the nested context language reuse features", December 2009
11. ABNT NBR Associação Brasileira de Normas Técnicas (2007) Digital terrestrial television standard 06: data codification and transmission specifications for digital broadcasting, part 2—GINGA-NCL: XML application language for application coding. São Paulo, SP, Brazil, November 2007. http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Ing_2008.pdf
12. ITU-R Recommendation BT-1699. Harmonization of declarative content format for interactive TV applications. Geneva, 2009
13. www.softwarepublico.gov.br
14. Ierusalimschy R, Figueiredo LH, Celes W (2006) Lua 5.1 reference manual, August 2006
15. W3C World-Wide Web Consortium (2003) Scalable vector graphics—SVG 1.1 specification, W3C recommendation. http://www/w3/org/TR/SVG11
16. Bulterman A, Dick CA, Rutledge A, Lloyd W (2009) SMIL 3.0—flexible multimedia for web, mobile devices and daisy talking books, 2nd edn. Springer, Berlin
17. Costa RMR, Moreno MF, Soares LFG (2008) Intermedia synchronization management in DTV systems. In: Proceedings of ACM symposium on document engineering (DocEng 2008), Sao Paulo, Brazil, pp 289–297
18. Cesar P, Bulterman DCA, Geerts D, Jansen J, Knoche H, Seager W (2008) Enhancing social sharing of videos: fragment, annotate, enrich, and share. In: Proceedings of ACM international conference on multimedia, Vancouver, Canada, October 2008. ACM, New York
19. ISO/IEC International Organization for Standardization 14496-1. Coding of audio-visual objects, part 1: systems, 3rd edn
20. Concolato C, Le Feuvre J, Moissinac JC (2007) Timed-fragmentation of SVG documents to control the playback memory usage. In: Proceedings of ACM symposium on document engineering, New York, USA
21. Onural L (2007) Television in 3-D: What are the prospects? Proc IEEE 95(6):1143–1145
22. ISO/IEC 19775-1.2. X3D architecture and base components, 2nd edn. International Organization for Standardization, July 2008
23. Soares LFG, Moreno MF, Sant'Anna F (2009) Relating declarative hypermedia objects and imperative objects through the NCL glue language. In: Proceedings of the ACM symposium on document engineering, Munich, Germany, September 2009

24. Gelernter D, Jagganathan S (1990) Programming linguistics: a first course in the design and evolution of programming languages. MIT Press, Cambridge
25. ISO/IEC 13818-6. Information technology—generic coding of moving pictures and associated audio information, part 6: extensions for DSM-CC. ISO Standard, 1998
26. Façanha R, Fonseca NLS, Rezende PJ (1999) The S2 piggybacking policy. Multimed Tools Appl 8(3):371–383
27. Bachelet B, Mahey P, Rodrigues RF, Soares LFG (2007) Elastic time computation in QoS-driven hypermedia presentations. ACM Multimed Syst J 12(6):461–478
28. Jeong T, Ham J, Kim S (1997) A pre-scheduling mechanism for multimedia presentation synchronization. In: Proceedings of IEEE international conference on multimedia computing and systems, Ottawa, Canada, pp 379–386