

Intermedia Synchronization Management in DTV Systems

Romualdo Monteiro R. Costa

Marcelo Ferreira Moreno
Catholic University of Rio de Janeiro
Rua Marquês de São Vicente 225
22453-900 Rio de Janeiro, RJ, Brazil
+55 21 3527-1500 Ext: 3503

Luiz Fernando Gomes Soares

romualdo@telemidia.puc-rio.br

moreno@telemidia.puc-rio.br

lfgs@inf.puc-rio.br

©ACM, (2008). This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the Eighth ACM Symposium on Document Engineering, {VOL1, ISBN 978-1-60558-081-4, (09/2008)}
<http://doi.acm.org/10.1145/1410140.1410203>

Intermedia Synchronization Management in DTV Systems

Romualdo Monteiro R. Costa

Marcelo Ferreira Moreno
Catholic University of Rio de Janeiro
Rua Marquês de São Vicente 225
22453-900 Rio de Janeiro, RJ, Brazil
+55 21 3527-1500 Ext: 3503

Luiz Fernando Gomes Soares

romualdo@telemidia.puc-rio.br

moreno@telemidia.puc-rio.br

lfgs@inf.puc-rio.br

ABSTRACT

Intermedia synchronization is related with spatial and temporal relationships among media objects that compound a DTV application. From the server side (usually a broadcaster's server or a Web Server) to receivers, end-to-end intermedia synchronization support must be provided. Based on application specifications, several abstract data structures should be created to guide all synchronization control processes. A special data structure, a labeled digraph called HTG (Hypermedia Temporal Graph) is proposed in this paper as the basis of all other data structures. From HTG, receivers derive a presentation plan to orchestrate media content presentations that make up a DTV application. From this plan other data structures are derived to estimate when media players should be instantiated and when data contents should be retrieved from a DSM-CC carousel or from a return channel. If the return channel provides QoS support, another data structure is derived from the presentation plan, in order to determine when resource reservation should take place. For content pushed by broadcasters, HTG is used in the server side as the basis for building the carousel plan, a data structure that guides the order and frequency that media objects should be broadcasted.

The paper's proposals were partially put into practice in the current open source reference implementation of the standard middleware of the Brazilian Terrestrial Digital TV System. However, this reference implementation is used just as a proof of concept. The ideas presented can be extended to any multimedia document presentation player (user agent) and content distribution server.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation - *languages and systems, markup languages, multi/mixed media, hypertext/hypermedia, standards.*

D.3.2 [Programming Languages]: Language Classifications - *specialized application languages.*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DocEng'08, September 16-19, 2008, São Paulo, Brazil.

Copyright 2008 ACM 978-1-60558-081-4/08/09...\$5.00.

General Terms

Design, Languages, Standardization.

Keywords

NCL, Temporal graph, digital TV, intermedia synchronization, middleware.

1. INTRODUCTION

Intermedia synchronization has been one of the most important QoS (quality of service) issues in document engineering, multimedia data communication and multimedia system management. The spatial and temporal synchronization among media assets plays a central role in all phases of a digital TV (DTV) application's life cycle: since its specification, passing through its playout stage, to its execution at the receiver.

In DTV applications, spatiotemporal synchronizations must deal with both predictable events¹ (like the end of a media-span presentation with known duration and known beginning time) and unpredictable events (like viewer interactions). Moreover, to maintain the application flow synchronized, content and content-presentation adaptations can be necessary. Such adaptations are usually performed during runtime and can be another source of unpredictability.

The use of the timeline paradigm for temporal synchronization is only appropriate in supporting "strongly coupled" datacasting, that is, data with predictable synchronization points, like an audiovisual data coming from a digital camera. MPEG-video [15] is an example of a coding process based on timeline.

In multimedia applications in which unpredictable events are common, the event-driven paradigm (also called constraint/causality paradigm) is usually the best conceptual model to guide all phases of an application's life cycle. Different from the timeline paradigm that binds synchronized media points to points in time, the event-driven paradigm bases its synchronization support on the relative spatiotemporal positioning of events, independent from when (the absolute moment in time) a synchronization would happen and even if it would happen.

The use of event-driven paradigm brings three main associated subjects:

¹ In this paper, an *event* denotes any occurrence in time with finite or infinitesimal duration

- How to specify spatial and temporal synchronizations among events;
- How to control the application's presentation in order to guarantee that its specified synchronization relationships will be respected; and
- How to manage transmissions from servers to receivers, maintaining the QoS required to assure a synchronized presentation in the client device.

As for the first subject, spatiotemporal event-driven synchronization can be specified using declarative or imperative languages.

Declarative languages emphasize the declarative description of an application, rather than its decomposition into an algorithmic implementation, as it is done when using imperative languages. Such declarative descriptions generally are a high-level specification, and thus they are easier to be designed than imperative ones, which usually require a programming expert. However, declarative languages typically have a focus on their design principles. When an application focus matches the declarative language focus, the declarative paradigm is mostly the best choice.

Declarative languages can be designed with a focus on the spatial and temporal synchronization among components of an application. Examples of so-called time-based declarative languages are NCL (Nested Context Language) [18], the standard language of the Brazilian Terrestrial DTV System [1], and SMIL (Synchronized Multimedia Integration Language) [22], a W3C Recommendation.

Imperative languages, in particular scripting languages, like ECMAScript [8], provide an expressive support for application development. However, they have some drawbacks for the content production: they usually require a programming expertise for application development; they occasionally put at risk the application portability; and the management of the application temporal flow (the presentation control) is much more difficult to be done as a rule, besides being more error-prone, since the application programmer is in charge of this task, in contrast with a time-based declarative language, whose language engine does it automatically.

Moving on to the second subject, the application's presentation control, upon receiving a document specification, with all spatial and temporal relationships among its objects defined, the language engine must try to guarantee the author's (the programmer's) descriptions. In order to support this task, several data structures are computed from the application specification. These data structures must represent all possible predictable and unpredictable events.

One of this data structures is what we call the *presentation plan*, responsible for supporting the presentation control. During a DTV application presentation, all information gathered from viewers and the receiver's system, 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 current state. This is a common situation found in a DTV environment where viewers are allowed to explicitly pause a DTV application and then to resume it at some later time — possibly days or weeks later, and even on a different device, still preserving all actions previously done.

Managing a multimedia presentation state control is to manage the associated DTV application's presentation plan. Besides allowing

changes on the plan traverse, thus making a dynamic presentation, it is also important to allow saving the data structure context, at least the current state of the plan navigation.

Several times, in order to maintain the synchronization, content and presentation adaptations are necessary. Therefore, in DTV applications, the presentation plan must also be able to represent the specification of adaptation points that can happen during the presentation flow.

Finally, to support the last mentioned subject, the transmission management, other data structures can be necessary both in the server and in the client side.

One of these data structures is what we call the *carousel plan*, which is computed from the application specification in the server side (usually a broadcaster's server). This data structure will be important to guide the management of DSM-CC carousels [12]² for pushed content.

Other important data structures are usually derived from the presentation plan in the receiver side. These new data structures are used to guide content prefetching (the *prefetching plan*) and QoS negotiation (the *QoS plan*) in the receiver's interactive channel (return channel), and to guide player instantiations (the *player-load plan*), necessary for the exhibition of media-object's content.

This paper discusses the aforementioned data structures, calling attention to their importance in managing system's resources in a DTV environment, usually a scarce asset in the receiver side, and in maintaining the spatiotemporal synchronization of DTV applications.

The paper's proposals were partially put into practice in the current open source reference implementation of the Ginga middleware [1], the standard middleware of the Brazilian Terrestrial Digital TV System. In Ginga, all mentioned data structures are based on temporal graph concepts. However, Ginga's reference implementation is used in this paper as just a proof of concept. The ideas presented can be extended to any multimedia document presentation player (user agent) and content distribution server.

The paper is organized as follows. Section 2 presents some related work. Section 3 discusses the presentation-plan building and use in the receiver side of a DTV system, in order to guide the application flow control. Section 4 discusses the building and use of the carousel plan in the server side, in order to guide the object carousel management. Section 5 introduces the use of the prefetching and QoS negotiation plans in the receiver side, in order to guide the pulled content transmission in the interactive channel. Section 6 is reserved for the final remarks.

2. RELATED WORK

Several players are available for Web-based multimedia applications. One of them is the Ambulant open-source player [7], which aims to be a reference implementation of the SMIL engine, at this time supporting SMIL 2.1 [22] specification.

The Ambulant document presentation begins parsing the document specification and building its DOM tree [21]. Each node in the tree

² DSM-CC carousels provides cyclical data transmission from a server to client receivers

that represents media assets (*image, text, audio* etc.) or SMIL composite elements (*par, seq, excl*) is controlled by a data structure called time node. Composite elements have temporal semantic embedded and group other elements (media assets or compositions, recursively). Edges in the tree preserve temporal relationships among composites and their children, defined by the composite semantics. Other relationships among time nodes are represented by semantic links, in addition to tree edges. The combination of both relationships forms the complete time graph [17] [22]. Each time node has an associated state machine, which is in a particular presentation state depending on the document presentation flow.

NCL user agent open-source implementation for Ginga middleware [1] is similar to Ambulant. NCL context (composition) nodes group media nodes and other context nodes, recursively. However, NCL context nodes have no predefined semantics. Instead, NCL context includes link elements defining spatiotemporal relationships among context's children. Similar to SMIL time nodes, each NCL node has an associated state machine, which is in a particular presentation state depending on the document presentation flow.

Both in NCL and SMIL, the presentation flow starts from a root node. From this moment on, relationships defined among composition's children are triggered in appropriate specified relative moments, changing presentation states of related nodes.

The data structure defined by NCL nodes and SMIL time nodes is sufficient and efficient enough for playing a document, if it is assumed that all media assets are available at the moment their presentation are triggered. However, in DTV broadcasting environments, where receiver's resources are normally restricted, it is impossible to have all media contents before the document starts. In this case, contents must be retrieved from a remote location during a document play. The aforementioned data structure is very inefficient in guiding this task. To derive the moment that a prefetching should be done, or that a QoS negotiation for a data transfer must be started, or that an object carousel should be loaded, usually requires the simulation of the document presentation flow, which is almost always very inefficient.

Aiming at to bypass this problem, the NCL user agent reference implementation builds another data structure based on a temporal graph data model, called Hypermedia Temporal Graph (HTG) [19]. HTG represents in a unique digraph structure all relationships among presentation states of all media assets that compose a document, instead of having this relationships distributed and embedded in composition elements. HTG represents all predictable and unpredictable events that can cause changes in the presentation state machines of media assets (which can be started, paused, resumed, stopped and aborted) and all content and content-presentation alternatives, as discussed in Section 3.

The CMIFed Player [20], responsible for controlling the presentation of CMIF documents, basis for the SMIL language specification, builds a graph of temporal dependencies at a compile phase, when media object relationships (parallel, sequential etc.) are processed and event expected times are calculated. The execution control uses an internal clock to traverse the temporal graph and to identify the moments to fire presentation actions. The system can simulate the document execution and locate presentation inconsistencies, like loops and resource contention. CMIFed also uses the graph information to perform a content prefetching whenever the presentation scheduler becomes idle. However, the

proposed ideas could not be directly applied to SMIL 2.1 specification, due to the complexity of SMIL temporal relationships.

HTG concept is similar to Firefly temporal chains [6]. In Firefly, the main temporal chain is built corresponding to a sequence of predictable presentation events, initiated from the event that corresponds to the beginning of the document presentation. As usual, a predictable event is one that can have its time of occurrence computed based on the occurrence time of another event. Other auxiliary temporal chain of events can also be computed, each one being initiated by an unpredictable event, like a viewer interaction, alternatives that can only be evaluated during runtime, etc. All events of an auxiliary temporal chain must be predictable in relation to another event present in the same chain.

During runtime, each auxiliary chain is joined to the main chain, as soon as its initial unpredicted event is evaluated. Therefore, only at the time the last unpredicted event of the document happens, the whole temporal chain can be obtained.

Different from Firefly's temporal chains, HTG is not a simple timeline of presentation actions and represents all possible presentations of the whole document. In addition, instead of adding branches (auxiliary chain) to the graph (the main chain), according to the document presentation flow, branches are pruned from the graph, when alternatives or unpredictable events cannot occur anymore. Firefly chains are used only to compute media content duration in order to maintain document temporal consistency.

A document presentation can run as many times as necessary, partially or totally. For each runtime, a new activation is done, without any historical relationship with the previous ones. However, there are many situations where the application's presentation history is important to be maintained, in order to reach the desired results, 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;
- When a viewer changes the TV channel, thus starting another application in the new channel, but then regrets and returns to the previous channel, resuming the application and inheriting all information previously given, all answers previously provided, all interactions previously done, all environment information previously set, etc.

These two examples illustrate different situations in a DTV environment. In the first one, the application, and thus all its contents, must be resumed from the exact point where it had been paused, or from a preceding point. In contrast, in the second one, the application must be resumed in a future point of the TV program, with regards the time it had been interrupted. Nevertheless, in both cases, all previous information should be preserved.

As far as the authors know, only HTG provides an efficient data structure to allow a document presentation resuming, considering all interactive actions and all alternative choices performed in the past.

Jeong et al. [16] have developed a mechanism for pre-scheduling multimedia presentations. The prefetcher, named EPS (*Event Pre-Scheduler*), estimates the maximum time for recovering the entire content of each document object and builds a prefetching plan in order to have all media contents ready at their playing time. The building algorithm postpones the start of the document presentation

as a whole, in order to avoid gaps and loss of synchronization during the presentation. The execution plan data structure, used as input for the prefetching plan calculation, is based on the partition of media objects in minimum segments that satisfy the Allen equality condition (*equals*) [2]. At document runtime phase, there is a monitor that compares the actual object prefetching duration with the expected duration previewed in the plan. If the actual duration overcomes the predicted one, the system runs an instant scheduling algorithm to recalculate the object presentation duration, in order to maintain the temporal segment synchronization. When necessary, the algorithm sacrifices static media objects, shrinking their durations. The interesting characteristic of such an algorithm is that it allows correcting delays introduced not only by the operating system kernel interruptions, but also by the algorithm itself.

EPS presents a practical approach for identifying several requirements that must be considered when elaborating prefetching mechanisms. Other contribution of this work is the implementation of a strategy for building the prefetching plan. However, the proposal only considers documents exclusively based on predictable relationships and predictable media object durations. Furthermore, all contents must be locally and contiguously stored in a hard disk, ignoring aspects related to network transmission. The strategy presented always prefetches the entire object content to memory, before start playing it. It is not possible for a media player to download data that is currently being presented in parallel with data that is being prefetched.

In order to maintain intermedia synchronization in networks that allow QoS negotiation, intramedia QoS negotiation (delay, bandwidth, etc.) can be used to guarantee that content pulled from an interactive channel will be available for presentation in the receiver side when they are necessary. None of the proposals discussed in this paper mention the intramedia QoS negotiation and maintenance issue, not even the NCL player reference implementation for Ginga. However, Ginga anticipates the future use of this procedure discussing how HTG can be used to derive the moment a QoS negotiation should be started, as presented in Section 5.

Focusing on the presentation scheduling and on pushed contents coming from servers, the MPEG-4 system [14] allows the specification of spatial and temporal synchronizations among events using a declarative language called XMT-O [14]. This language inherits several SMIL modules, allowing that relationships can be defined using compositions with temporal semantics or using events defined over media objects. However, another MPEG-4 language called BIFS (Binary Format for Scenes) [14] is used to control the presentation, from servers to receivers. This approach is implemented in the MPEG-4 reference player [13] and others, like the OSMO4 [11].

BIFS follows the timeline paradigm³ and its specifications can be synchronously multiplexed with media objects into a stream [14] that can be presented at the same time it arrives, avoiding previous storage. Although BIFS follows the timeline paradigm, it simulates some content presentation adaptation and also viewer interactions.

³ Besides timeline, BIFS optionally supports some temporal constructs using FlexTime model [14]. FlexTime allows defining synchronization relationships using a very limited set of Allen relations [2].

Using BIFS, multiple object contents can be simultaneously transmitted and the content to be presented can be chosen at presentation time. BIFS also supports a limited type of viewer's interactions which can change the value of the object visibility property. Additionally, an interactive action can also change the entire MPEG-4 document being presented.

The use of the timeline paradigm of BIFS can facilitate the presentation control, since tasks like player loading and content transmission can be driven by absolute time positions on the time axis. The main problem, however, is the limitation to represent spatial and temporal synchronizations among events, mainly the unpredictable ones. In MPEG-4, the conversion from XMT-O to BIFS preserves the occurrence times of predictable events but the semantics of relative relationships among events is lost.

XHTML-based languages, such as BML [3], ACAP-X [4] and DVB-HTML [9] allow a declarative description of relationships involving unpredictable events; in fact, only viewer interactions. Intermedia synchronization in its broad sense can only be achieved using imperative coding, commonly written in ECMAScript [8]. DOM events and external events⁴ can trigger a scripting object used to provide temporal and spatial synchronization, content and presentation adaptability, and other facilities that are not otherwise possible. DTV middleware systems based on these languages offer no support for prefetching, QoS negotiation or carousel management. In these middleware systems, only intermedia synchronization support for presentation is considered, and even though, by means of imperative language-based control.

An alternative to represent content synchronizations in the aforementioned DTV middleware systems is to split a document, which represents a whole TV application, into several small documents fired along the time by commands issued by authors. This approach can only be used for simple applications and although it eventually avoids the disadvantages of using imperative languages for synchronization purposes, the application's logical semantics is completely lost. Note that this approach indeed uses a timeline paradigm where the absolute moments in time are defined by commands sent by application authors.

However, the split of a document into several small documents is frequently used as an alternative for avoiding previously storage of contents in receivers. This alternative is commonly adopted in DTV middleware systems to allow receivers with limited resources. Instead of using a solution similar to MPEG-4 BIFS, pushed contents are transported inside DSM-CC object carousels [12]. The carousel management is very simple in this case: small DSM-CC carousels for small split documents are created and transmitted. The split documents are fired by using DSM-CC stream events, and they need to be transmitted only until their corresponding stream events are dispatched. However, the responsibility of splitting an application efficiently and generating the corresponding stream events is passed to the application authors. This can be a very difficult task and more prone to errors.

An autonomic management of object carousels will need a data structure for guiding content insertion and exclusion. As an additional advantage, the original application does not need to be

⁴ In DTV systems, it is usual to have DSM-CC stream events [12] triggering the execution of imperative codes.

split, maintaining the logical semantics of all relationships among its media contents. As far as this paper authors know, only Ginga middleware provides support for carousel management, as discussed in Section 4.

3. INTERMEDIA SYNCHRONIZATION FOR PRESENTATION CONTROL

Authoring goals are very different from presentation goals, as regards data structures for spatial and temporal synchronization definition. Specification languages usually intend to offer high-level constructs to aid the authoring process. These constructs favor relationships among media objects that exist in the author's mind model (logical semantics). Presentation data structures are closer to the execution machine and should offer low-level primitives in order to make easy the presentation scheduling.

Actually, preserving the high-level abstraction, or a data structure close to it, can make difficult the presentation flow control, as discussed in the previous section with regard to NCL and SMIL node structures.

The presentation phase involves the management of media players and the delivery of media contents to them. Usually, a user agent is composed of a set of players, each one responsible for the exhibition of specific media types. It is important to have each specific player instantiated and ready before the exact moment in time its corresponding media presentation should start, in order to avoid undesirable delays, mainly present when receiver's resources are restricted, like processing power and memory.

When the time for an object presentation is reached, the corresponding player must receive the media content to be presented. Moments in time for player instantiations and for starting presentation are both derived from the presentation data structure and correspond to the *player-load plan* and *presentation plan*, respectively.

DTV environments have some specific characteristics that must be taken into account when defining an efficient presentation data structure. When an application does not have any semantic relationship with the main audiovisual streams, it may be presented from its start point, as soon as the TV channel is tuned in. However, when the main audiovisual streams are part of an application, the application must start from the current audiovisual runtime position. 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.

When DTV applications are started, interactive events that might have happened in the past, like viewer interactions, should be taken in account. An acceptable and simple solution consists in ignoring all past interactions that might have happened until the chosen starting point. This approach is possible for an application that is running for the first time. However, it is not applicable in several cases, for example, if the application has been paused and then restarted. In this case, interactive events that can be inferred from the last running should be maintained. Thus, maintaining the presentation state is another requirement for the data structure we are searching for.

The presentation structure of Ginga middleware was developed in conformance with these requirements. Both the player-load plan and the presentation plan of the Ginga reference implementation are

derived from a directed graph, which is built at document compile time.

NCL recognizes three types of events: presentation event, corresponding to playing a content anchor (whole media-object content, or part of this content); selection event, corresponding to a viewer interaction (selection of a content anchor); and attribution event, corresponding to setting a value to a property (variable). Each event defines a state machine that should be maintained by the receiver user agent. An event can be in the sleeping, occurring or paused state, and change its state upon receiving actions: start, stop, pause, resume and abort.

The Ginga presentation data structure, called Hypertext Temporal Graph (HTG) [19], is composed of vertices, which represent actions (for state changes) performed on event state machines, and directed edges that represent relationships among actions, derived from the document specification in NCL. An edge is labeled by a condition that must be satisfied in order to trigger the action specified in the edge's output vertice.

HTG maintain all relationships among actions, predictable or unpredictable, depending on the condition type associated with its edges. HTG defines simple and compound conditions. A simple condition is defined by a temporal interval that must be spent to fire the edge traverse (and so the action defined in the edge's output vertice), or by a variable that must be evaluated in relation to a desirable value, or still by external actions, such as viewer interactions. Compound conditions are defined through logical operators (or, and, not) binding two or more conditions.

The two last mentioned simple condition types are used to represent unpredictable events and adaptation points. A variable is evaluated during runtime and a viewer interaction is always unpredictable within a certain time interval.

After defining an application starting point⁵, HTG can be used to derive the presentation plan. When applications contain only predictable events, graph edges are labeled only by temporal intervals. From the document starting time, the graph traverse identifies every action that must be applied to media players. These actions can have their moment in time computed taking into account the time intervals required to satisfy conditions from the HTG entry point to the corresponding action vertices. This set of actions and corresponding moments in time compose the presentation plan.

While the presentation plan for applications without unpredictable events can be entirely computed a priori (at compile time), for applications where unpredictable events may occur this is not true. However, the same procedure previously applied can be used to compute actions and their corresponding moment in times for all predictable events from an application starting point to an unpredictable event; and from each unpredictable event to the next unpredictable event in the graph traverse. In this last case, the compute moments in time will be relative to the moment in time that the starting unpredictable event of the traverse path happens.

During an application presentation, as soon as an unpredictable event time is known, the presentation plan is updated changing all moments in time relative to this event to be now relative to

⁵ NCL does not restrict its documents to have only one entry point. The language defines that each port element (anchor) in a composition can be a document entry point.

document starting time. That is, by adding the moment in time that the unpredictable event has occurred to moments in time relative to this unpredictable event.

Sometimes an unpredictable event does not happen within the time period at whatever time its occurrence is allowed. In this case all actions in the presentation plan whose occurrence depends only on this event must be removed.

The presentation plan, together with HTG and all local and global variable settings made during a document presentation, represent the past and possible future history of an application. This information is called the presentation state of an application and is the only information needed for starting or resuming an application from whatever point.

Considering memory limitations of DTV receivers, media players should only be instantiated when necessary. For example, they usually cannot stay instantiated after being used, waiting for a possible next utilization. However, due to resource limitations too, the time needed to instantiate a media player can introduce a delay long enough to cause loss of temporal synchronization among content presentations. In this case, a player-load plan should be computed from the presentation plan.

For each start and resume presentation action in the presentation plan, new moments in time must be computed for the equivalent player-load plan, taking into account the delay for each player instantiation. Other types of actions present in the presentation plan must be disregarded when building the player-load plan.

From the presentation plan, the prefetching plan and the QoS negotiation plan must be derived, as discussed in Section 5.

4. INTERMEDIA MANAGEMENT OF BROADCASTING CONTENT

In terrestrial DTV systems it is usual to have some application data transmitted in synchronized streams using timestamps (following the timeline paradigm), as for example, the main audio and the main video. It is also usual to have other data (other video, audio image, text, etc) transmitted asynchronously (without timestamps) in other streams. Although these other objects are transmitted asynchronously, they can be synchronized with themselves and with data coming in synchronous streams. In this case, the event-driven paradigm is used for synchronization, with another asynchronously transmitted object, the DTV application, carrying the synchronization specification, as mentioned in Section 1.

In terrestrial broadcasting system, all streams are multiplexed and transmitted to receivers inside a frequency channel. In the client side, receivers must be tuned in a desired channel to start receiving its contents.

Since a channel can be tuned at any time and because asynchronous transmitted data must be delivered previously to its exhibition time, the data must be sent repeatedly, in order to guarantee its reception independent from the time a channel is tuned in. MPEG System standard defines a cyclical data structure to transport these data called DSM-CC carousel [12]. Following this standard, data may be put in a carousel that is transmitted time after time in an elementary stream. Using object carousels data objects may be inserted more than once and in any place within a carousel.

The carousel bit length, the carousel stream transmission rate, and the space between instances of a same object give the maximum

delay for retrieving that object. The carousel transmission rate is limited by the TV channel bandwidth. Moreover, as this bandwidth is shared with other data streams synchronized by timestamps, including the main audiovisual streams, if the carousel time-length increases, the audiovisual quality of service decreases. Thus, it is very important to maintain the carousel as small as possible.

When a receiver is tuned in a desired channel, the main audiovisual streams can be immediately presented. However, the asynchronously transmitted data already received must wait the application specification arrival to know the exact time (calculated by the presentation plan) to be presented. Thus, the application specification must have the shortest possible delay. As a consequence, it should be inserted in a carousel more than once, as explained before.

The simplest way to deal with the carousel management is having a carousel with all data (application specification and its media contents) needed to run an application. In this case, the server side has a very simple procedure. The receive side has two extreme cases of operation, as follows.

The first case requires initiating an application only after receiving all its data. This can generate a large delay that can impair the correct application running, especially if there are temporal relationships involving asynchronously transmitted data and other data streams, for example, the main video stream. This approach also usually requires a large storage capacity in the receiver, what is uncommon.

The second case works without needing the storage of all application's media contents. In this case the receiver must assure that, after starting an application, it will always have a media content already extracted from the carousel when the moment in time for the content presentation arrives. For a correct performance, the receiver engine should build from the presentation plan, discussed in Section 3, a *prefetching plan*, specifying the moment a media object should be extracted from the carousel to the system memory, as discussed in Section 5.

The two previous cases assume that a carousel transports all information needed to run an application. As aforementioned, this is not a good approach, since it leaves a smaller bandwidth for the main audiovisual streams, decreasing their QoS. Therefore, it is worth to try to work with carousels containing only part of applications. This presumes that the server side knows which part of an application a carousel should transport in a certain moment.

One solution, adopted in some related work presented in Section 2, is to pass the carousel management responsibility to the application author. The author must split the application 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, the author is also responsible for triggering these applications in precise moments, using another DSM-CC facility: stream events [12].

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

In the Brazilian Terrestrial DTV System (SBTVD), DSM-CC object carousels are used. The carousel plan is built based on the HTG model, but in the server side this time.

The carousel plan is similar to the presentation plan built in receivers, with the exception that, once built, it does not need to be updated, since there is no feedback coming from receivers. Therefore, as for the server knowledge, all unpredictable events must be treated as if they will happen at the moment they will be enabled.

From the carousel plan, the server-side middleware should estimate which objects must be placed inside the carousel, how many times, in which places, and which objects must be removed from; a difficult optimization problem indeed.

In the current server reference implementation for SBTVD, the removal processes is done in its plenitude, but the insertion needs a better algorithm. In the current implementation, the application object is always present in the carousel. It is the only object inserted several times, in order to minimize the application starting delay. Other objects are inserted in the carousel depending on its length, the carousel length, their maximum retrieving delay, and their expected presentation times, obtained from the carousel plan.

It should be noted that the use of an optimized carousel does not relieve receivers from managing the carousel loading. If they do not have sufficient memory to retrieve the whole carousel data, a prefetching plan should be built to guide the retrieval, as discussed in the next section.

5. INTERMEDIA MANAGEMENT FOR CONTENT LOADING

As discussed in Section 3, based on the presentation plan the user agent knows when media contents must be played. However, in order to be able to execute this task, not only media players must be instantiated but also media contents must be available on time.

The player-load plan, discussed in Section 3, guides the player instantiation to solve the first requirement. In order to deal with the second requirement, two approaches are possible, as already discussed for data obtained from DSM-CC carousel. The same procedures can be extended for pulled data obtained by using return channels (or interactive channels): i) requiring all application's contents before starting it; ii) requiring application media contents on the fly, that is, during the application execution.

As aforementioned, the first solution requires a large receiver's storage capacity, which is usually impossible for low cost receivers. Moreover, such solution can introduce an unbearable application starting delay.

The second solution is much better, but demands from receivers the control of content retrievals. For pushed data transported in DSM-CC carousels, a prefetching plan must be built, establishing when a media object should be taken out of a carousel.

A carousel prefetching plan is built based on the presentation plan and the estimated carousel delay. In the current Ginga middleware reference implementation, a very simple procedure is used, based on the worst case carousel delay. In the procedure, all unpredictable events are assumed to happen immediately after they are enabled.

As for pulled data, the download procedures depends on if the interactive channel network allows intramedia QoS negotiation or not. Intramedia QoS deals with single media objects. It is associated with the moment that contents are obtained from storage locations, the network transfer rate, transfer delay and transfer jitter, and scheduling policies in the involved (clients' and servers') operating

systems. In fact, intramedia QoS is an important support in order to guarantee a correct maintenance of intermedia synchronization.

If QoS support is not provided by an interactive channel, receivers must download media objects guided by a prefetching plan. This plan is built based on the presentation plan, as usual, taking into account the estimate network transfer delay and transfer jitter for each object. Because the plan is built based on estimations, prefetching mechanisms in receivers' middleware will be useful only to minimize the probability of having temporal mismatches. 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's contents before starting an application can be considered a special case of this solution. When building the prefetching plan, a conservative approach assumes that all unpredictable events happen immediately after they are enabled.

In interactive channels that offer QoS support a better control can be achieved. In this case, from their presentation plans, receivers build their QoS plans, taking into account the transfer delay and the transfer jitter that will be negotiated for each object. The QoS plan is then used to trigger resource reservation procedures in order to obtain the desired QoS. If the negotiation is succeeded, then it is guaranteed that the media object will be in a receiver on time; else a new negotiation can be started with more relaxed QoS parameters or a new negotiation can be started in future time, but, in this case, with more strict QoS parameters. Using QoS, the chances for temporal synchronization mismatches are reduced.

QoS negotiation in DTV systems brings back the interesting topic about resource reservations in advance raised by mobile computing work. Resource reservation in advance enables scheduling and allocation of resources at an early stage in time. This way, the availability of resources can actually be guaranteed for the point in time when the resources are needed.

In mobile computing, handoffs can cause QoS breaks. Thus resource reservation in advance should be made based on the next future location of a mobile device. The problem in mobile computing is to know the future location and when it will be reached. So, QoS in advance could be achieved only based on estimations and with resource waste. In contrast, DTV systems do not have these constraints.

Based on the presentation plan, the exact moments in time to make 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 even more the mismatch probability, since it enlarges the time range for resource reservation negotiation.

The reference implementation of Ginga does not provide support for QoS negotiation yet. In receivers, besides the presentation plan, only the prefetching plan is built using a conservative algorithm. It is assumed that all future unpredictable events will happen immediately after they are enabled, and the worst case transfer delay and jitter (both for the DSM-CC carousel and for the interactive channel) are assumed. In the implementation, the interactive channel is continuously probed, adjusting the worst case delay and jitter on the fly.

6. FINAL REMARKS

This paper presents an enhanced set of data structures to support intermedia synchronization management in client and server sides of a DTV system. The proposed data structures are derived from a directed time graph model (HTG), which preserves not only event execution times but also all temporal relationships among events, including unpredictable relationships, such as viewer interactions and those requiring on-the-fly content adaptations.

All proposed data structures take into account that applications should run in receivers where resources are normally restricted, and that applications should have a good response time.

The HTG model can also be used in the authoring phase. The Composer authoring tool [10] uses HTG to guide its temporal view exhibition. In Composer's temporal view, a graphical representation of all application's media objects is presented in a relative time axes, making easier the creation of temporal relationships among media objects.

The temporal graph structure and most of the other derived data structures discussed in this paper were implemented in the reference implementation of the Ginga middleware, the standard middleware of the Brazilian Terrestrial Digital TV System, used in this paper as a proof of concept. This open source implementation can be obtained from www.ncl.org.br.

In the current implementation, algorithms used for establishing the moments in time for the carousel plan creation and for the prefetching plan creation are very simple and are based only on the worst case, as mentioned in the paper. They need to be enhanced in near future.

Another future work is to use adaptations for compensating delays greater than the predicted values used to build the plans, stretching and shrinking media object presentations. NCL DTV profiles do not support the specification of variable content-presentation duration, as does the NCL full profile. However, NCL user agents can make small adjustments, mainly in static media presentations. The ideas presented in [5] could be used in a future implementation, as well as in a future DTV profile specification.

7. ACKNOWLEDGMENTS

The authors would like to thank Carlos Salles and Marcio Moreno who provided a thoughtful discussion of this work, tracked down and also fixed problems in the initial reference implementation of Ginga.

8. REFERENCES

- [1] 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/A_BNTNBR15606-2_2007Ing_2008.pdf.
- [2] Allen J.F. 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11), November 1983, pp. 832-843.
- [3] ARIB Association of Radio Industries and Business. 2004. ARIB Standard B-24 Data Coding and Transmission Specifications for Digital Broadcasting, version 4.0, 2004.
- [4] ATSC Advanced Television Systems Committee. 2000. ATSC Data Broadcasting Standard - A/90, 2000.
- [5] Bachelet, B.; Mahey, P.; Rodrigues, R.F.; Soares, L.F.G. 2007. Elastic Time Computation in QoS-Driven Hypermedia Presentations. *ACM SIGMM Multimedia System Journal*, vol.12, No. 6. Springer Verlag. (Maio de 2007); pp.461-478. ISSN: 0942-4962
- [6] Buchanan M.C., Zellweger P.T. 1992. Specifying Temporal Behavior in Hypermedia Documents. I Proceedings of European Conference on Hypertext (Milan, Italy, December 1992). ECHT'92.
- [7] Bulterman D.C.A., Jansen J., Kleantous K., Blom K., Benden D. 2004. AMBULANT: A Fast, Multi-Platform Open Source SMIL Player. In Proceedings of ACM International Conference on Multimedia (New York, USA, 2004).
- [8] ECMA International - European Association for Standardizing Information and Communication Systems. 1999. ECMA – 262 – ECMAScript Language Specification. 3rd Edition. <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.
- [9] ETSI European Telecommunication Standards Institute. 2006. ETSI TS 102 812 V1.2.2 Digital Video Broadcasting “Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.1.1”.
- [10] Guimarães, R.L.; Costa, R.R.; Soares, L.F.G. 2008. Composer: Authoring Tool for iTV Programs. In Proceedings of European Interactive TV Conference (Salzburg, Austria, July 2008). EuroITV 2008.
- [11] GPAC Project on Advanced Content. OSMOSE Player for MPEG-4 – OSMO4. <http://gpac.sourceforge.net/player.php>
- [12] ISO/IEC International Organization for Standardization 13818-6. 1998. Information technology – Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC.
- [13] ISO/IEC International Organisation for Standardisation. 14496-5. 2000. Coding of Audio-Visual Objects – Part 5: Reference Software. 2nd Edition, 2001.
- [14] ISO/IEC International Organization for Standardization 14496-1. 2004. Coding of Audio-Visual Objects – Part 1: Systems. 3rd Edition.
- [15] ISO/IEC International Organization for Standardization 14496-10. 2005. Information Technology – Coding of Audio-Visual Objects – Part 10: Advanced Video.
- [16] 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, 1997), pp. 379-386.
- [17] Schmitz P. 2001. The SMIL 2.0 Timing and Synchronization Models: Using Time in Documents. Technical Report. Microsoft Research.

<http://research.microsoft.com/research/pubs/view.aspx?type=Technical%20Report&id=439>

- [18] Soares L.F.G., Rodrigues R.F. 2006. Nested Context Language 3.0 Part 8 – NCL Digital TV Profiles. Technical Report. Departamento de Informática da PUC-Rio, MCC 35/06. <http://www.ncl.org.br/documentos/NCL3.0-DTV.pdf>.
- [19] Soares L.F.G., Costa, R.M.R; Moreno,M.F. 2008. Graph-Based Schedulers for Resource Management and Presentation Control in a QoS Architecture for DTV Applications. Technical Report. Departamento de Informática da PUC-Rio, MCC 12/08.
- [20] van Rossum G., Jansen J., Mullender K.S., Buterman D. 1993. CMIFed: A Presentation Environment for Portable Hypermedia Documents. In Proceedings of ACM Multimedia (Anaheim, USA, August 1993).
- [21] W3C World-Wide Web Consortium. 2004. Document Object Model – DOM Level 3 Specification. W3C Recommendation.
- [22] W3C World-Wide Web Consortium. 2005. Synchronized Multimedia Integration Language – SMIL 2.1 Specification, W3C Recommendation. <http://www.w3.org/TR/2005/REC SMIL2-20051213/>