Just-in-time personalized video presentations

Conference Paper · September 2012

DOI: 10.1145/2361354.2361368

CITATIONS
11

READS
94

4 authors:

Jack Jansen
Centrum Wiskunde & Informatica
91 PUBLICATIONS **1,729** CITATIONS
SEE PROFILE

Pablo Cesar
Centrum Wiskunde & Informatica
292 PUBLICATIONS **2,509** CITATIONS
SEE PROFILE

Rodrigo Laiola Guimarães
Universidade Federal do Espírito Santo
60 PUBLICATIONS **278** CITATIONS
SEE PROFILE

Dick Bulterman
Vrije Universiteit Amsterdam
222 PUBLICATIONS **3,580** CITATIONS
SEE PROFILE

Just-in-Time Personalized Video Presentations

Jack Jansen[1], Pablo Cesar[1], Rodrigo Laiola Guimaraes[1,2], Dick C.A. Bulterman[1,2]

[1]CWI: Centrum Wiskunde & Informatica, Amsterdam, NL [2]VU University, Amsterdam, NL

jack.jansen@cwi.nl, p.s.cesar@cwi.nl, rlaiola@cwi.nl, dick.bulterman@cwi.nl

ABSTRACT

Using high-quality video cameras on mobile devices, it is relatively easy to capture a significant volume of video content for community events such as local concerts or sporting events. A more difficult problem is selecting and sequencing individual media fragments that meet the personal interests of a viewer of such content. In this paper, we consider an infrastructure that supports the just-in-time delivery of personalized content. Based on user profiles and interests, tailored video mash-ups can be created at view-time and then further tailored to user interests via simple end-user interaction. Unlike other mash-up research, our system focuses on client-side compilation based on personal (rather than aggregate) interests. This paper concentrates on a discussion of language and infrastructure issues required to support just-in-time video composition and delivery. Using a high school concert as an example, we provide a set of requirements for dynamic content delivery. We then provide an architecture and infrastructure that meets these requirements. We conclude with a technical and user analysis of the just-in-time personalized video approach.

Categories and Subject Descriptors

D.3.2 [**Language Classifications**]: Specialized application languages; I.7.2 [**Document and Text Processing**] Document Preparation - Languages and systems.

General Terms

Design, Experimentation, Standardization, Languages.

Keywords

Video mashups, late binding of media, seamless playback.

1. INTRODUCTION

When considered as a document, a video is distributed as a compressed, read-only object. During production, the video is assembled from available source content during the editing phase. It is then compressed for storage and transfer efficiency, and made available as an on-line community video as a block downloadable media item, or incrementally as a streaming media object. The nature of video shifts any significant amount of content personalization to the creation phase: if viewer A and viewer B are interested in different content aspects, they traditionally require separate video objects.

One approach to providing more personalized presentations of a single event is to supply a few video feeds in parallel, giving the viewer an opportunity to select a vantage point of interest. This is an approach already in use for live sports events, and it could be applied to other public events, both live and pre-recorded, such as popular concerts. Given an appropriate user interface, a viewer could take on the role of a director and select the shot of greatest personal interest. This approach is limited by the number of cameras used and the transfer bandwidth available.

Our work considers another approach to video personalization. We examine a model in which many cameras are available as sources (which may cover only short sections of the event, and more than feasibly could be broadcast in parallel) and in which a viewer is offered the ability to select a particular view based on not only camera positioning, but also on personal affinity with the content of the video. We consider a high school music concert in which parents record fragments of the event. By combining all of the media objects captured and then analyzing the contents, we allow an interested viewer to obtain a personalized presentation that focuses on individual performers or instruments. The main motivation to watch this content is not the musical quality of the event, but the strong personal bonds that exist with the performers. These bonds are highly partitioned: in general, the family and friends of, say, the trombone player will be interested primarily in content related to this person, while other performers (such as the clarinet player) may have an essentially disjoint community of interested viewers. This means that for each sub-community, separate personalized presentations will be required. Even within a sub-community, viewers will have different interests (and tolerances) to the concert content: some will want to see all of the footage involving their performer of interest, while others will be more than satisfied with a few key fragments. In essence, this shifts content selection within a video object to the consumption phase of video viewing. We refer to this approach as *just-in-time content personalization,* since the media content to be shown is determined by the context of the viewer and his/her interaction while viewing the media.

This paper focuses on the language and real-time processing required to support dynamic just-in-time content personalization. We consider a video to be an incremental electronic document in which follow-on fragments are determined dynamically. While other work describes the selection of appropriate source material based on user preferences and media annotations [23] or the general application of dynamic video selection [8], this paper concentrates on the infrastructure needed to dynamically assemble and render the presentation as a seamless video document.

DocEng'12, September 4–7, 2012, Paris, France.

The contribution of this paper is an architecture that enables the presentation of dynamic just-in-time media content. We describe the requirements for such a system, the processing architecture needed for client-side selectivity, and our implementation experience. We also summarize the results of user tests that help determine the viability of our approach.

This paper is structured as follows: section 2 provides an overview of our user-case scenario and the requirements for our work. Section 3 gives a brief overview of approaches that enable end-user customization of video content. Sections 4 and 5 describe the design and implementation for our approach. Section 6 describes the results of our work, including both a technical and user analysis of our results. Section 7 concludes with thoughts for broader applicability and future work.

2. SCENARIO AND REQUIREMENTS

During the past decade, the wide introduction of video capture devices has enabled video to become a ubiquitous method for documenting shared events. At the same time, any one video has not really become a ubiquitous method of presenting a shared events because it provides only a single point of view: that of its creator.

2.1 Classifying Community Events

In this paper, we focus on support for relatively personal events. In particular, we study asynchronous communication and scenarios in which groups of people are motivated to edit and share audiovisual material over extended time periods. Unlike many collaborative editing systems, the primary purpose of the content sharing is not the publishing of a collective common work, but the provision of highlighting the roles of individuals within a shared event.

Our use case, which we call *MyVideos*[1], centers around a musical performance of young high school students. At the concert the audience consists of parents and friends of the performers, many of whom capture the event using video. After the concert, all video material is collected, preprocessed for temporal alignment, analyzed for appearance of performers, instruments and shot types and stored in a database called the *Vault*. Once this has been done, the material is made available to the interested parties for browsing, sharing and viewing through a specialized MyVideos web application, shown in Figure 1. The MyVideos application allows access to recorded material in a number of different ways, ranging from a rather traditional browse and assemble interface to having a presentation generated fully automatically by the system.

In this article we are interested in the latter view option, called the *Interactive Narrative*. The system provides an initial baseline presentation based on the viewer's preferences and relationship to the performers. This baseline presentation can then be altered - while watching - based on user input.

A viewing sequence starts with an initialization phase in which a particular viewing session contacts the *Narrative Engine*, requesting an initial playlist of video fragments and audios. The design and implementation of the narrative engine are out of the scope for this article, they can be found in [23], here we provide a summary useful for understanding the contribution of this paper. The Narrative Engine has access to metadata for all available clips, including performers and instruments featuring in each clip, esthetically pleasing in and out points and cinematographic shot type such as wide angle or closeup. In addition, the Narrative Engine has a profile for the user, so it knows which performers or instruments the viewer is interested in. This data is fed into a rule-based engine which compiles a story from the relevant clips. The resultant compilation is sent back to the client application, which is in charge of rendering it to the viewer.

2.2 Requirements for Just-In-Time Videos

The narrative engine provides the web-based user client with a dynamic playlist of video clips that are refined continuously based on user feedback. From this perspective of this paper, our problem can be formulated as: *can we enable an external agent to do late composition of audio and video material while still maintaining a seamless playback experience for the viewer?* This problem can be broken down into two distinct requirements:

- combining audio and video material from multiple sources in such as way that it plays back identical as it would have done had the combination been done statically, and
- ensuring that the material for the next shot is available sufficiently in advance so that we can satisfy the previous requirement.

There is also a key interaction requirement:

- Through a series of user evaluations and user studies, we have determined that most viewer want a *lean-back* control experience, in which they provide broad guidance on interests rather than a *lean-forward* experience, where the user actively browses and selects individual items.

To maintain the lean-back experience within the context of the first two requirements, the interactions will not have an immediate effect (such as a channel change or switching to a different YouTube video would have), but they will influence the material selected in the near future. Figure 2 shows a screenshot of video playback, with two interaction buttons. Note that having the buttons on-screen is a choice made for this implementation. In a production system it would be better to have the buttons on a secondary screen, or at least make them less intrusive. This is purely a presentation choice, it makes no difference to the structure of the application.

3. BACKGROUND AND RELATED

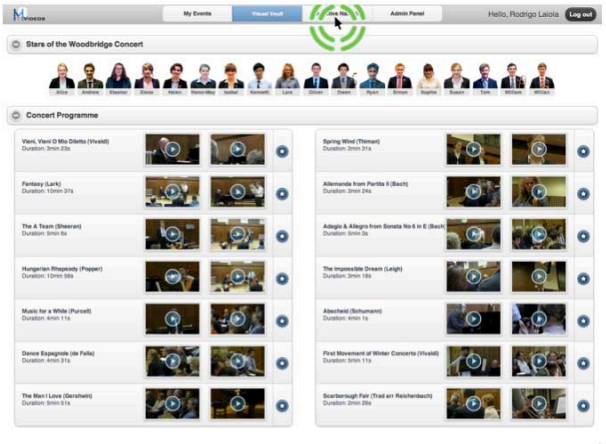

Figure 1. MyVideos web application user interface.

[1] MyVideos is a use case with the FP7 Project TA2: Together Any- where, Together Anytime. See: http://ta2project.eu

WORK

The contribution of this paper is a document model and a design that allows for just-in-time video presentations. In a sense, this work on be video as an on-demand instance of a *video mash-up*: a custom video projection of a shared event based on community assets. The primary difference of our approach is that we shift the customization phase of the mash-up production to the client side at viewing time, rather than to a source side at production time. In order to provide a general context for our work, this section reviews related research in the areas of community-based mashups and rich multimedia document models.

3.1 Community-Based Storytelling

Research on community-based video mashups has primarily focused on content analysis for searching purposes. Kennedy and Naaman [11] describe a system for synchronizing and organizing user-contributed content from live music events, creating an improved representation of the event that builds on the automatic content match. Shrestha et al. [17], report on an application for creating mashup videos from YouTube recordings of concerts. They present a number of content management mechanisms (e.g., temporal alignment and content quality assessment) that then are used for creating a multi-camera mashup. Results indicate that the final videos are perceived almost as if created from a professional video editor. Unlike our work, however, the resulting mashup is not dynamic but a static conventional media object. (Multiple objects are required to capture alternative story lines.) Shaw et el. [16], explore the creation of videos by remixing existing assets in a database, focusing on the human aspects and reporting on how final videos were created. More recent work has proposed a media sharing application that takes into account the interpersonal ties. This tool is capable of producing presentations shows based on events, people, locations, and time [18]. In comparison to our work, these applications do not use a client-side facility for compiling stories during viewing.

Various AI approaches for interactive storytelling have been suggested in the past decade. One representative example is Vox Populi [1], in which rhetorical documentaries are created from a pool of media fragments. More recently, a system capable of creating different story variants from a baseline video was presented and evaluated [14]. In comparison to our work, these articles only focused on the quality of the final story (in narrative terms), without taking into consideration the underlying document model, the performance, and the rendering issues. Based on our design guidelines, the work reported in this article is applicable to any third-party narrative engine, enabling a robust delivering and rendering environment.

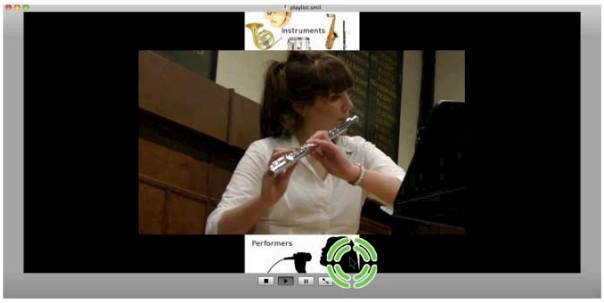

Figure 2. Video Playback, with interaction

3.2 Mashups in Multimedia Documents

Structured multimedia documents provide support for mashups, as the composition of sequential video fragments with an external audio track. Such models were traditional rather static, in which the media items to be included were known in advance not meeting the requirements impose by our problem space. Recently, additions to the standards are intended for extending the flexibility and dynamism of the models, as we will detail in this section.

NCL (Nested Context Language) [19] is the standard XML-based application language for defining interactive multimedia presentations in the Brazilian Terrestrial Digital TV System (SBTVD-T). In NCL, authors can take advantage of its high-level constructs to describe, in a declarative manner, the temporal behavior of a multimedia presentation. Authors can as well associate hyperlinks with media objects, define alternatives for presentation, and describe the layout of the presentation on multiple devices. Moreover, NCL provides support for imperative scripts in order to extend its computational power [20]. Unlike HTML and Flash, NCL has a strict separation between the document's content and structure, and it provides non-invasive control of presentation linking and layout. This means that an NCL player can be used to render (portions of) videos in the context of a video mash-up, and to control the timing and rendering properties of in such a way that potentially the videos can be displayed seamlessly.

While some investigations have been reported on dynamic content insertion, support for generalized dynamic modifications of a running presentation are restricted. Some works target for dynamism by allowing commenting and annotating user generated videos [5], but without changing the actual video being played. Other works [4] propose a method for live editing of NCL programs, preserving not only the presentation semantics but also the logical structure semantics defined by a professional author. In their solution a void media placeholder was inserted in the presentation, which then could be modified in real-time by a third party. We follow a similar method in our work, but go one step further since in our case we cannot predict the time at which the presentation ought to be modified, nor the number of elements and their duration.

SMIL (Synchronized Multimedia Integration Language) [2] is another example of a structured multimedia document model that can be used a content container within our scenario. Similarly to NCL, SMIL provides temporal and synchronization constructs for the creation (and adaptation) of complex multimedia presentations. The inclusion of SMIL State [9,10] provided support for dynamic variables in running multimedia presentations. In our work we take advantage of such functionality showing it usefulness for rather complex scenarios.

Document engineering research has investigated the problem of dynamically generating and adapting documents. The terminologies come in different flavors depending on the target domain: just-in-time, real time, live editing etc. For example, King et al. [12] presented a set of document extensions that may dynamically react to continuously varying inputs, both in a continuous way and by triggering discrete, user-defined, events. Such extensions are discussed and realized in the context of SMIL Animation and SVG, but could be applied to many XML-based languages. Our work differs from the one described in [12], whose main target is animation. Also related to our work, Zhang et al. [22] provided a general solution to supplementing virtual

documents from third party applications with just-in-time hypermedia support, utilizing dynamic regeneration, re-identification and re-location. One key difference is the fact that in our case the document temporal model plays a major role, as we focus on temporal operations that do not destroy the running timegraph of the presentation.

3.3 Mashups in Video Sharing Systems

In YouTube (and other commercial systems), dynamic mashups are not supported. End-users have to find suitable source material, cut it into shots, and assemble an encoded final video. While this solution does not impose hard requirements on delivery and rendering, it is limited in terms of adaptability and user interaction.

One solution might be the use of playlists in such environments, which are typically supported using Flash or HTML5. However, since the smallest granularity of a playlist is an entire video, mash-ups are only possible if, and only if, the videos are processed in such a way that the end point of the previous video matches the start point of the following one. Unfortunately, YouTube does not allow for seamless playback, since for each video of a playlist the Web page is reloaded. In other words, the application envisioned in this article is not possible to be implemented in existing commercial video sharing systems.

4. *MyVideos* CLIENT DESIGN

The design of the MyVideos application was driven by the three requirements sketched in section 2. The application is segregated into three major components: the MyVideos web interface, the Narrative Engine and the playback component. The Narrative Engine is implemented as a web server, where the other two components send their requests. There are basically three types of requests: *get-first-playlist*, *get-new-playlist* and *user-interaction*. The first two return a playlist that contains information on the media required within the following 20 seconds (approximately). The third one does not return any information, but is used to inform the Narrative Engine that some user interaction has happened, which the engine can then use to modify the playlist returned in the subsequent get-new-playlist call.

The playlist is explained in more detail in section 5, here it suffices to note that it does not contain absolute playout timing information, only an ordered list of media clips, in-points and out-points. Layout and presentation information is also not part of this playlist. This isolates the Narrative Engine from the details of multimedia composition, scheduling and playback, these are now the responsibility of the playback component.

The central aspect of this work is supporting the playback quality requirement in section 2: we need a playback engine that can seamlessly play back audio and video segments that come from different sources. Seamless switching from one video clip to the next is a difficult problem, especially if the clips come in over the network, and the in-point for the destination clip need not be on an I-frame. A complete description of the problem and solution for the case when the media items are known a priori can be found in [6, 7]. To summarize here, seamless switching requires:

- a mechanism to open, seek and prefetch clips before they are needed,
- scheduled control over switching from one clip to the next.

Taken together, these requirements point in the direction of a solution that uses a *timegraph*. Languages that treat media items as isolated timed islands in a sea of scripting, such as HTML5, do not allow the tight control from the second requirement.

From [7] we can learn that using SMIL as the multimedia format addresses the quality requirement, but for our dynamic use case this introduces a new problem: we now need to dynamically update the active SMIL document with information that is not known a priori. Moreover, we need to do these dynamic updates in such a way that we maintain the seamless playback. Please note that although we say here that this problem is introduced by our use of SMIL this does not mean that the choice of SMIL is a bad one: if we had chosen HTML5 as the basic format we would have faced a similar problem that was only different in the details, because in HTML5 the functionality of the timegraph will have to be encoded in a procedural scripting language.

When we modify the media items in the SMIL presentation, we need to make sure that our modifications do not affect the timegraph structure. In [10], the authors have presented a taxonomy of editing operations on active multimedia documents. We reproduce the table from that document here as Table 1. If we apply this taxonomy to our requirements it should be clear that operations from the *Selection* and *Adaptation* clusters are not sufficient for the problem at hand. Therefore, we need to come up with a design that uses operations from the *MediaItem* cluster only, and specifically operations from that cluster that do not lead to timegraph modifications. It turns out that we we cannot add (or remove) media nodes from the timegraph, we cannot modify `begin`, `end` or `dur` attributes of existing media nodes, we can only modify `src`, `clipBegin` and `clipEnd` of existing items.

These observations have led to a design with a carrousel-like structure in the SMIL presentation, with the individual media items in the carrousel being taken from SMIL State objects. The SMIL code now simply loops over the carrousel continuously. Now we only need to design one more bit of functionality: obtaining playlist data from the Narrative Engine and inserting it into the SMIL State objects, and communicating user interaction to the Narrative Engine. This is handled by the *Carousel Assistant*. It communicates user interactions to the Narrative

Table 1 - Taxonomy of Document Transformations ([10]**)**

Selection cluster	Selection among predefined media items. It does not affect the time graph of the presentation.
Adaptation cluster	Modification of the style or layout composition. They do not affect the time graph of the presentation.
MediaItem cluster	Modification of the content of a media item. They might affect the time graph of the presentation.
Structural cluster	Modification of the temporal composition (add/remove item). They must recompute the temporal graph.

Figure 3. Flow of the presentation.

Engine as they occur, and periodically contacts the Narrative Engine to obtain new playlists. The data from these playlists is then inserted into the SMIL State variables.

The user-perceived temporal flow of the presentation is outlined in figure 3. The pre-concert section is a short set of clips meant to communicate the atmosphere of the concert: people greeting friends and finding seats, instruments being setup, etc. Not coincidentally, we also present the user with the initial set of interaction buttons during this pre-concert section. Close to the end of the pre-concert section we disable the interaction buttons and request a playlist from the narrative engine, which we then present as the first concert section. Again, close to the end of the concert section we disable the interaction (if any), and request the next playlist. This continues until the narrative engine decides to wrap things up in the conclusion section. This is played and the presentation ends.

The overall structure and the data flow is sketched in Figure 4. (1) shows how user interaction is communicated to the Narrative Engine. (2) shows the Narrative Engine selecting clips for playback. (3) and (4) show how media clip references pass from the Narrative Engine to rendering. (5) shows how the media data flows. Note that the Carousel Assistant is always the client (active agent) in the communication with the Narrative Engine. This is in line with the requirement that the Narrative Engine has no realtime requirements. In an earlier prototype we had the Narrative Engine responsible for actively delivering new playlists to the playback engine, which meant it did have limited realtime requirements. This led to problems when the playlist did not arrive in time.

5. IMPLEMENTATION

In this section we will examine the implementation of the client components of Figure 4. This includes an extension of the Ambulant Player[2] and a Carousel Assistant implemented in Python.

For the SMIL player we have selected the Ambulant Player. The player is an open source implementation of a SMIL 3.0 compliant player. The player includes a parser for the XML file, an scheduler for deciding when nodes become active and inactive, a layout manager for compositing different media items, and a DOM tree that stores the logical structure of the document. Moreover, Ambulant provides native media renderers for handling the media items and uses datasources for retrieving the actual data (e.g., video) included in the presentation. Ambulant is extensible, allowing us to explore complex scenarios, like the one presented in this paper.

By extending Ambulant, a number of requirements for just-in-time multimedia mashups have been met. More specifically:

- *Sequential ordering*: SMIL has a temporal construct (in particular `<seq>`) for supporting playback of dynamically generated playlists.
- *Synchronization*: there is a need to synchronize different media elements (`<par>`). In particular, interaction components (such as buttons) need to be synchronized with active video elements

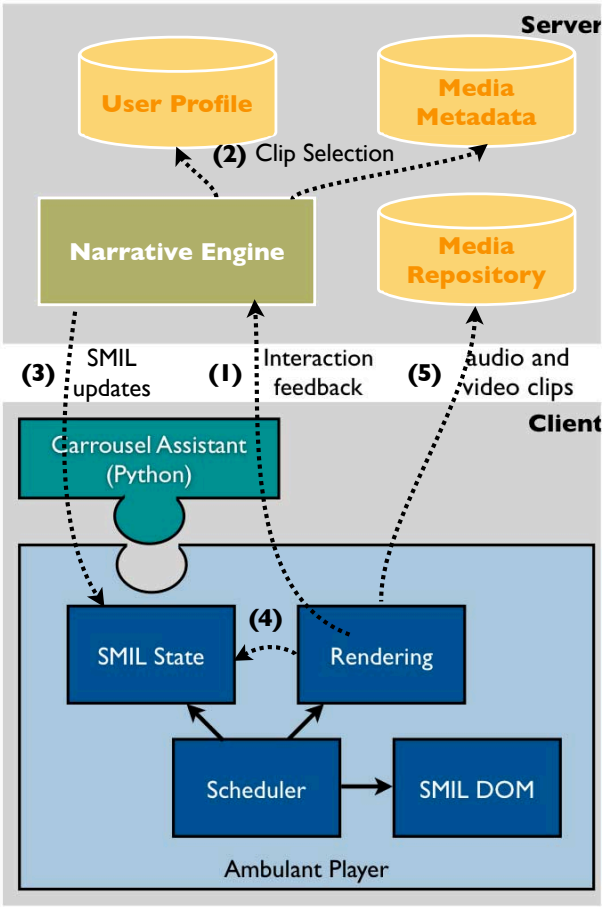

Figure 4. Architecture, with data flow.

[2] http://ambulantplayer.org/

during playback. By using SMIL, the narrative engine does not need to provide hard synchronization values (timestamps), but only relative synchronization structures (activate this interaction 2 seconds after such video has started).

- *Interaction*: the use case requires interaction capabilities, so users can interact in real time, while watching a story. In this case, XML interactive elements (`<submission>`) are supported by the rendering engine.
- *Seamless cuts*: there is a need to provide seamless playback of media items for offering an immersive experience to the user. In this case, the rendering engine supports elements such as `<prefetch>`, that have been extended for media fragments rendering.

In addition, a number of practical considerations are addressed:

- *Interfaces*: the presentations need to for interface to an external engine. In this case, the rendering engine needs to interface with the narrative engine (Carousel Assistant), which will dynamically provide the most adequate playlists for being rendered.
- *Dynamism*: the application depends on dynamic changes on presentations while they are active. In particular, the rendering engine supports SMIL State for managing changes without affecting the overall performance of the presentation, and without breaking the active timegraph.

The basic structure of the SMIL document is outlined in Figure 5. The document heavily relies on SMIL State for allowing dynamism. SMIL State allows for variable-like structures that will not be evaluated until they become active, allowing us to

```
<smil>
  <head>
    <state xmlns=''>
      <ta2>
        <videoplayed>0</videoplayed>
        ...
        <playlist>
          ...
          <videoItem>
            <url>http://.../clip1.mp4</url>
            <begin>5.2s</begin>
            <end>7.2s</end>
          </videoItem>
          ...
        </playlist>
        <prefetch>
          <prefetchItem>
            <url>http://.../clip4.mp4</url>
            <begin>22.0s</begin>
            <end>24.0s</end>
          </prefetchItem>
        </prefetch>
        ...
  <body>
    ...
    <seq repeatCount="indefinite">
      ...
      <par>
        <video src='{playlist/videoItem[4]/url}'
          clipBegin='{playlist/videoItem[4]/begin}'
          clipEnd='{playlist/videoItem[4]/end}'/>
        <prefetch
          src='{prefetch/prefetchItem[4]/url}'

clipBegin='{prefetch/prefetchItem[4]/begin}'
          clipEnd='{prefetch/prefetchItem[4]/end}'/>
      </par>
      <setvalue name="videoplayed" value="4"/>
```

Figure 5. Relevant sections of SMIL document

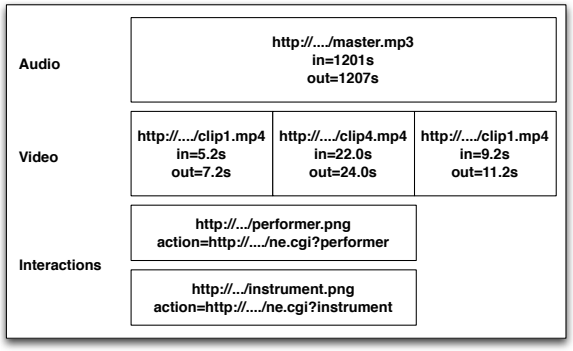

Figure 6. Playlist data as generated by Narrative Engine.

instantiate them as the Narrative Engine send subsequent playlists compilations.

The state of the document includes information about the current element that is played, links to which interactions at a given moment will make HTTP requests, and instantiations of the parameters of the videos (and audios) that compose the presentation. All these values (and the number of them) are not known at the time of creating the SMIL document, but will be filled in by the Carousel Assistant by interacting with the Narrative Engine.

The body of the document is a carousel: an infinite loop of video and audio structures. Since the number of videos that compose a compilation is unknown, we have created a list-like structure that will iteratively be repeated until the compilation is finished. In this structure, we include the video fragment to be played and a prefetch instruction for the next video fragment. The particular values are stored in the SMIL State section and being dynamically updated. This general structure is robust in terms of timing: (1) we assure that video fragments will be played sequentially and (2) we address the requirement of seamless playback (QoE). The body includes as well the audio elements (not shown in Figure 5), which use a similar structure as the video elements (item + prefetch next item). In the scenario, the audio to be be played is not the one embedded in the video file. Different cameras (and in different positions) will create perceptible audio changes, thus disturbing the quality of experience. In this case, SMIL allows for parallel constructs for synchronizing audio and video fragments. Moreover, interactions (not shown in the Figure) can be as well synchronized with specific video clips, thanks to the SMIL language.

Finally, `<setvalue>` allows the body of the document (rendering engine) to communicate with the Carousel Assistant. When a video structure has been played, a SMIL State variable (`videoplayed`) is updated, which triggers a function of the Carousel Assistant.

We have implemented the Carousel Assistant in Python. The Ambulant Player support extension modules in either C++ or Python, and for the task at hand Python seemed like the obvious choice. Ambulant extension modules have access to the complete internal API of the Ambulant Player, but because we have chosen to stick to the MediaItem cluster of operations (as per [10]) we only use the APIs to access the SMIL State portion of the document. This will guarantee that we do not modify the timegraph, which is one of the requirements for maintaining seamless playback.

The main entry points into the Carousel Assistant are SMIL State callbacks. The SMIL document is set up in such a way that it modifies SMIL State variables every time something interesting happens: (a) a video element has finished playback, (b) an audio element has finished playback, or (c) user has done an interaction.

The Carousel Assistant has requested callbacks for changes on any of these variables. However, as the operations that need to happen on the basis of these events may involve communicating to the Narrative Engine the operations are not executed inline, but instead a second thread is signaled, which will then implement the operation. The main thread immediately returns to the SMIL player so it can continue processing without further delay.

For video and audio elements finishing, the Carousel Assistant checks how many more elements are left in the currently playing playlist. Depending on this, it decides whether to contact the Narrative Engine for a new playlist (when the previous to last item has been played), otherwise it does nothing. The Narrative Engine returns the next playlist as a JSON data structure. The contents of this data structure are sketched in Figure 6. The data structure includes a complete new playlist with an audio fragment, a set of video fragments, and a number of interactions. Consecutive data structures (playlists) form the full interactive story the Narrative Engine generates.

The audio fragment and the video fragments provide the source (URL) together with the in and out points; the interactions include information about the representation (image), the synchronization parameters (relative to the videos), and the action (as HTTP requests to the Narrative Engine).

The Carousel Assistant implements an infinite buffer, in which all these data structures are stored. When a new playlist (as in Figure 6) is received, it calculates where the the elements should be updated in the SMIL State section (it knows the number of videos played so far, and the length of the playing playlist). The values included in the JSON data structure update the following video items, audio items, and interaction items ensuring a concatenation

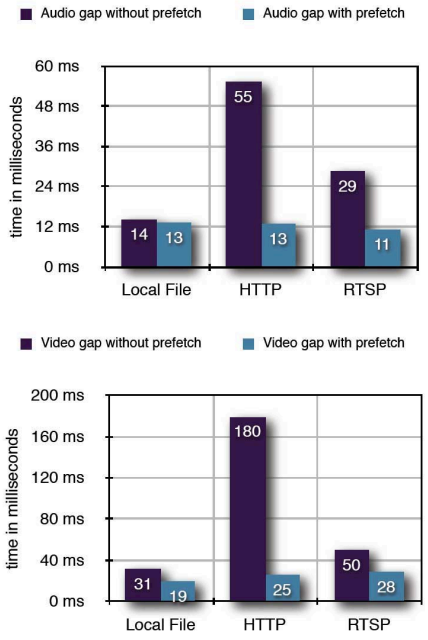

Figure 7. Delays for switching between clips (from [6]**)**

of playlists, and that these are played seamlessly from the end-user perspective.

For interaction events the Carousel Assistant has to do very little: the Narrative Engine has earlier supplied all the data it wants in case of a user selection and this data has been recorded in the SMIL State (action of the interaction). When an interaction event happens the assistant simply communicates this data to the Narrative Engine through an HTTP request. Nothing is returned: the Narrative Engine will use this data to update its internal data structures, which will influence the next playlist it supplies to the Carousel Assistant.

The implementation presented in this section supports complex and rich media rendering (and interactions). The complexity, in comparison to previous work, is based on the high-level of dynamism enforced by our scenario. It supports synchronization of video fragments, interactive elements, and audio components. Moreover, the number of playlists that will be rendered (and their duration) is unknown in advance, so void media placeholders are of little use (we do not know how many of them we will need). Finally, each playlist is composed of an audio element, an interactive element, and an unknown number of video elements. As a result a highly flexible and dynamic implementation, as the one reported in this section, is needed.

6. RESULTS

The work described in this article is evaluated from three different perspectives: seamless playback performance, dynamic composition performance, and the user experience. First, we evaluate the playback aspects of the system, showing that at playback time the delays between consequent video items is minimal. From the document model angle, we provide a set of timegraphs that show how our decision of using a rich multimedia model provide a robust solution for complex multimedia applications, meeting the timing and synchronization constraints required for the use case. Finally, we explore the QoE of our implementation. User evaluations of our application indicate high user satisfaction.

6.1 Seamless Playback

In terms of performance, the use case described in this paper is highly demanding, since media transitions between fragments need to be seamless. First, audio files need to be lip-sync with incoming video clips (even more when dealing with a music concert). Second, video clips do not necessarily start from the beginning (`clipBegin`), which requires a seek operation within the videos. Third, the sequence of video clips need to be played seamless, as if it was a pre-compiled video.

Previous research, more fully described in [6], has investigated the behavior of our playback engine with predetermined media items. Figure 7 shows clip switching delays from [6], here we are interested in the case of delivery over HTTP: delays as low as 13 ms for audio and 25 ms for video,. These are almost non-perceptible.

6.2 Dynamic Composition

The timing and synchronization constraints of just-in-time multimedia mashups are relatively high, when compared to previous works in rich multimedia document models [15]. The complexity resides on the dynamic and non-predictive nature of the application. Typically, multimedia document models allow for compositing media items with variable duration coming from different sources. This allows the rendering of synchronized

Figure 8. Visualisation of the runtime activity of a just-in-time multimedia mashup

media items (e.g., videos + extra material), which are reactive to user interaction. Nevertheless, the media items to be included in the application are known when packaging the presentation. More advanced research [4] explores the possibility of dynamically changing pre-existing media placeholders, thus allowing for live-editing of multimedia documents.

In our case, the constraints go one step further. The requirement here is to provide a robust solution that allows for dynamic changes in a running multimedia presentation. These changes are applied to active elements of the presentation (the video being played) and are not predictable. We cannot know in advance when the new playlists are to be inserted, we do not know their length, and we do not know the number of videos that they will contain. First, we have to composite media items from different sources in a synchronized manner (video + audio + interactions). Second, we need to provide as well support for lip-sync synchronization between these video clips and the audio track.

The results of the previous subsection are for statically known media items, so the next step is to show that our carrousel-based dynamic composition model does not introduce extra delays. In the process of determining these delays we realized that this is actually an instance of a much more general problem: if a multimedia presentation exhibits a different runtime scheduling pattern than what is expected, the author or maintainer needs to determine what causes this different behavior. This is because the

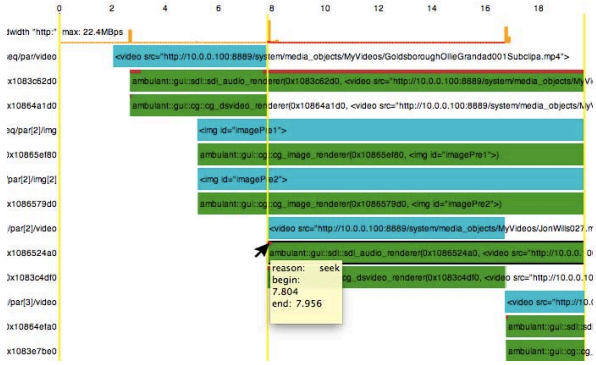

Figure 9. Visualising reason for presentation stall

reasons can be manyfold: mis-specified synchronization requirements, network overload, decoding overload, etc.

We decided to create a tool for multimedia playback that operates similar to what a multithreaded profiler does for programmers: show relevant execution data in a visual way to allow the author to quickly determine what the problem is. The tool is implemented as a plugin that is started the moment our multimedia player starts a presentation. The tool is reactive to dynamic modifications to the running presentation, as it shows the current internal state of the player. It visualizes various playback aspects at rendering time: bandwidth usage, timing and synchronization information, and current status of the XML elements (active, non-active).

To facilitate the visualizer, we instrumented the Ambulant Player to provide performance data. A raw performance data item consists of

- begin and end time,
- event type such as “element playing” or “element playback stalled”,
- parameters such as bandwidth consumed, or reason for stall.

These raw events are sent, in realtime, to the visualizer running in a browser window. Figure 8 shows an overview of the visualization after a MyVideos document has been playing for eight seconds[3]. We can see all the information regarding playback, including scheduling, bandwidth use and SMIL State information. Note that we see actual behavior, including unintentional delays and such. We can see the behavior of the different elements in the presentation (darker color indicates that an element has been started). In addition, we can see peaks on bandwidth allocation due to just-in-time prefetching, and red bands denote unintentional stalls. The user can interact with the data, for example by selecting items to see why it stalls, to hide certain node types, etc.

As figure 8 shows the visualization of an actual MyVideos run that does not stall we have also created a MyVideos document without the prefetch elements to facilitate seamless playback. Playback of this document incurred a 0.2 second stall during the first transition. Figure 9 shows the user investigating this stall.

The use of the visualizer tool allows us to demonstrate that we can maintain seamless playback characteristics in documents that use dynamic composition.

6.3 User experience

In addition to evaluating the seamless playback capabilities offered by our prototype, we have evaluated the user experience in field trials with real users. In order to evaluate our prototype implementation, a set of parents from a high school in Woodbridge (UK) has actively collaborated with this research. Parents (together with some researchers) recorded a school concert of their kids in November 2011. The concert lasted around 1 hour and 20 minutes, in which 18 students performed in 14 songs. A total of twelve cameras were used to capture the concert. The master camera was placed in a fixed location, front and sideway to the stage, set to capture the entire scene (a ‘wide’ shot) with no camera movement and an external stereo microphone in a static location. Eight cameras were distributed among parents,

[3] The PDF version of the paper in the Digital Library allows zooming to see more detail in the image.

relatives, and friends of performers. Members of the research team used the other 3 cameras. In total about 331 raw video clips were captured, some of which were recorded before or after the event.

Between January and February 2012, a subset of parents was invited to evaluate our prototype application with the material recorded in that concert. Nine people (from five families) participated in the evaluation. The participants consisted of performers, parents and other relatives of the teenagers that performed in the Woodbridge school concert. All were English speakers and living in the UK. Five participant (~56%) were 40+ years old; the other four people were in the 11-20-age range, three of which performed in the concert. While the number of users might be considered as too small for drawing conclusive results, the set of users represents a realistic sample for our use case. Moreover, all the parents attended the school concert and recorded the video material that was used during the evaluation. We strongly believe that in this case realism balances group size, so results are valuable, significant, and representative.

We used multiple methods for data collection, including interaction with the prototype application, followed by questionnaires. Each session started with a brief description of the component, and participants were instructed to describe their experiences. Based on our observations, responses to the questionnaires, and analysis of the collected audio/video material from the interviews, we discuss the results and findings (see Figure 10).

Overall, participants liked the MyVideos application (Q01) and indicated it helps recalling memories of past social events (Q02). A majority of subjects (eight out of nine) also reported the system makes them feel more connected to the people featured in the videos (Q03). It is important to mention that even though all participants liked the system, this does not imply they would pay for MyVideos (Q04). Among the justifications users point out it would depend on the cost and frequency of use. These positive results, while valuable, could be due to the novelty of using a system that gives a better viewing experience that just playing a plain single video stream (which also would mean that we did a good job). The rest of the section is dedicated to analyze the subjective reactions and comments from the users, providing a more complete discussion on what users thought about the MyVideos application.

In particular, participants described the interaction with the Interactive Narrative component as relaxing and enjoyable.

> *It is more than just getting into that concert again. It was doing something completely different, almost like doing another activity. In itself, it was a fun thing to do... it is not just a utility software to give you the best impression of the concert, it allows you to have some fun as well*. (Father of a performer)

In general, users mentioned the productions were visually compelling and had a good selection of cameras. Along these lines, one participant described the rich experience he had:

> *What I liked about that (Interactive Narrative) was that it started with a produced version and it almost felt we've gone up a level... it was more interesting somehow... .*

and added

> *I liked the variety of cameras... when the shots came from there (another direction) it was really interesting because I was not even there (on that side of the audience). On TV they seem to switch every few seconds... and (with Interactive narrative) you are getting a similar experience to what you are used to... I felt this a little bit more immersive (than the Vault).*

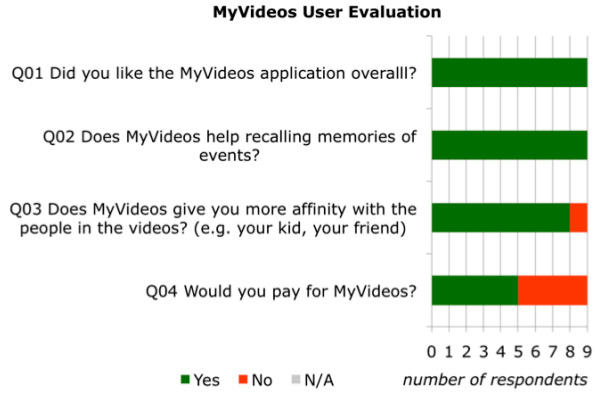

Figure 10. Results of the MyVideos evaluation.

In addition, some users indicated this was a 'lazy' experience in which they would mainly sit and watch, and not interact often.

> *It is like the red button on TV... I've never clicked on it... but I would like (to use it).*

Regarding the logical structure of narrative compilations, some users argued the playback of the pre concert material was disruptive and long. One participant suggested having a song selection interface before beginning the Interactive Narrative experience. Participants also complained about the lack of feedback in the user interface when they clicked on a button. This was a limitation of our implementation, which we plan to address in future work. In other occasions users mentioned that other forms of interaction also would be useful.

> *Some time ago the teacher said my daughter had problems with the (piano's) foot pedal... in a case like that it would be useful if we could say I am more interested on that specific part of the screen than any other shots of her*. (Father of a performer)

Based on observations we also noticed that even though users understood their choices (e.g. give me more of this performer) would be realized later in the presentation, in some cases they were interested in immediate changes (e.g. switch camera angles). In future work we plan to further investigate how these different deadlines for interaction/updates impact the timegraph structure.

7. CONCLUSIONS AND IMPLICATIONS

Video sharing is a powerful mechanism for recording and documenting shared events. From our work it is clear, however, that simply providing access to a collection of content on-line is in and of itself not sufficient to support a personalized view of that event: such personalization required dynamic tailoring of content based on the context of view at the time the media is accessed.

As reported here, our attempts at just-in-time personalization provide a powerful model for content delivery. Support for this model requires careful engineering and adequate networking support. Both of these are technical issues that can be straightforwardly addressed. A more problematic aspect of this work is that -- from a practical perspective -- it requires a

temporally-aware container document to host a skeleton incremental content document. In this respect, it is disappointing that recent trends in media deployment have favored a temporally-unaware model. Work with HTML5 [13] has demonstrated a desire to further compartmentalize (and thus marginalize) media content to become unstructured appendages to static document content.

In the future, lessons in including temporal synchronization will need to be integrated within the static models used by HTML. In this direction, INRIA has implemented Timesheets.js [3], a solution that takes advantage of HTML5 and CSS3, complementing them with a SMIL-based scheduler for handing timing and synchronization. Such implementation is based on a previous W3C working draft [21], aiming at compound XHTML documents, where different declarative languages could be used for the creation of complex multimedia applications. While we believe such work is in the right direction, research needs to be invested (as the exploration proposed in this article) for a solution that allows for dynamic modifications of running media items such as videos.

8. ACKNOWLEDGEMENTS

The work reported in this paper was funded in part by the European Community's Seventh Framework Programme under grant agreement No. ICT-2007-214793

9. REFERENCES

[1] Bocconi, S., Nack, F. and Hardman, L. 2008. Automatic generation of matter-of-opinion video documentaries. *Journal of Web Semantics*, 6(2): 139 – 150.

[2] Bulterman, D.C.A. et al. 2008. Synchronized Multimedia Integration Language (SMIL 3.0). W3C. URL=http://www.w3.org/TR/SMIL/

[3] Cazenave, F., Quint, V., and Roisin, C. 2011. Timesheets.js: when SMIL meets HTML5 and CSS3. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 43-52.

[4] Costa, R., Moreno, M., Rodrigues, R., and Soares, L.F. 2006. Live editing of hypermedia documents. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 165-172.

[5] Fraga, R., Motti, V.G., Cattelan, R.G., Teixeira, C.A.C., and Pimentel, M.G.C. 2010. A social approach to authoring media annotations. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 17-26.

[6] Gao, B, Jansen, J, Cesar, P and Bulterman, D. 2010. Beyond the playlist: seamless playback of structured video clips. *IEEE Transactions on Consumer Electronics*, (53)3:1495-1501.

[7] Gao, B., Jansen, J., Cesar, P., and Bulterman, D.C.A. 2011. Accurate and low-delay seeking within and across mash-ups of highly-compressed videos. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 105-110.

[8] Guimaraes, R.L., Cesar, P., Bulterman, D.C.A., Zsombori, V., and Kegel, I. 2011. Creating personalized memories from social events: community-based support for multi-camera recordings of school concerts. In *Proceedings of the ACM International Conference on Multimedia*, pp. 303-312.

[9] Jansen, J., and Bulterman, D.C.A. 2008. Enabling adaptive time-based web applications with SMIL state. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 18-27.

[10] Jansen, J., Cesar, P., and Bulterman, D.C.A. 2010. A model for editing operations on active temporal multimedia documents. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 87-96.

[11] Kennedy L., and Naaman. M. 2009. Less talk, more rock: automated organization of community-contributed collections of concert videos. In *Proceedings of the Interactional Conference on WWW*, pp. 311-320.

[12] King, P., Schmitz, P. and Thompson, S. 2004. Behavioral reactivity and real time programming in XML: functional programming meets SMIL animation. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 57-66.

[13] Pfeiffer, S. 2012. *The Definitive Guide to HTML5 Video*. Springer.

[14] Piacenza, A., Guerrini, F., Adami, N., Leonardi, R., Porteous, J., Teutenberg, J., and Cavazza, M. 2011. Generating story variants with constrained video recombination. In *Proceedings of the ACM International Conference on Multimedia*, pp. 223-232.

[15] Sadallah, M., Aubert, O., and Prie, Y. 2011. Component-based hypervideo model: high-level operational specification of hypervideos. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 53-56.

[16] Shaw, R., and Schmitz, P. 2006. Community annotation and remix: a research platform and pilot deployment. In *Proceedings of the ACM International Workshop on Human-Centered Multimedia*, pp. 89-98.

[17] Shrestha, P., de With, P.H.N., Weda, H., Barbieri, M., and Aarts, E.H.L. 2010. Automatic mashup generation from multiple-camera concert recordings. In *Proceedings of the ACM International Conference on Multimedia*, pp. 541-550.

[18] Singh, V.K., Luo, J., Joshi, D., Lei, P., Das, M., and Stubler, P. 2011. Reliving on demand: a total viewer experience. In *Proceedings of the ACM International Conference on Multimedia*, pp. 333-342.

[19] 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

[20] Soares, L. F. G., Moreno, M. F. and 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*, pp. 222-230.

[21] Vuorimaa, P., Bulterman, D.C.A., and Cesar. P. 2008. SMIL Timesheets 1.0. *W3C Working Draft*.

[22] Zhang, L., Bieber, M., Millard, D. and Oria, V. 2004. Supporting virtual documents in just-in-time hypermedia systems. In *Proceedings of the ACM Symposium on Document Engineering*, pp. 35-44.

[23] Zsombori, V., Frantzis, M., Guimaraes, R.L., Ursu, M.F., Cesar, P., Kegel, I., Craigie, R., and Bulterman, D.C.A. 2011. Automatic generation of video narratives from shared UGC. In *Proceedings of the ACM Conference on Hypertext and Hypermedia*, pp. 325-334.