Multimedia Presentation Integrating Media with Virtual 3D Realistic Environment Produced in Real Time with High Performance Processing

Arthur Pedro de Godoy [1], Caio César Viel [1], Erick Lazaro Melo [1], Diego R. C. Dias [2], Luis Carlos Trevelin [2], and Cesar A. C. Teixeira [1]

***Abstract*—Sophisticated interactive animation may be an interesting element to compose a multimedia document. However, potential demand for high performance computing can make this option impractical in digital interactive TV and mobile device environments, due to computing power restrictions of such platforms, especially when one consider the processing of complex 3D environments with a high degree of realism. In previous work, we proposed a solution to overcome such restrictions based on video streaming. Moving forward, in this paper is presented a solution that is more independent of platform, but still based on the same principle of video streaming. Each module can be specialized in different ways to solve problems related to the computer environment, such as network restrictions and latency on user interaction. Also in this paper is described how to take advantage of media-agnostic feature of some multimedia presentation machines to integrate this type of complex 3D media into multimedia documents. As a proof of concept, we extended and tested an NCL presentation engine to add support to this new type of media.**

Keywords*—*High-Performance Media, Multimedia Presentation Machine, Remote 3D Processing, Virtual 3D Environment, Video Streaming, Multimedia.

I. Introduction

Multimedia presentations composed by different types of media can enrich the user experience when watching, listening and interacting with them. An interesting type of media to be considered is interactive animations with a high degree of realism and complexity, which produces audiovisual information according to user's interactions. In this paper this type of media, which usually requires real time and high performance processing to be produced, is called *High Performance Media (HPM)*.

For example, consider a TV show based on an interactive expedition to a tourist spot, such as *Machu Picchu*. The application could be composed of a main video presenting an overview of the site and an HPM object representing a realistic 3D model of *Machu Picchu*. By means of the HPM object, users could explore areas and objects that were not explored or highlighted by the overview. The HPM object would be synchronized with the video of the overview. When the reporter moves to another scenario, the application would move too. It would be even possible for the user to meet the TV crew while navigating through the virtual environment. In addition, another media in the application, as high-resolution images or environment sound, could be added and synchronized with the user experience on HPM object. When the user enters a particular room, a specific song could start playing, for example.

Another interesting application could be a virtual contest between residents of different cities in a variety show. In addition to the audio and video of the TV show, with a television presenter, participants and audience attendance, the application could also include an HPM object offering a virtual environment with competitions and events to viewers. Some viewers of the competing cities could be awarded with the opportunity to participate in the contest, competing from their homes, while the others could follow the contest by choosing different visions of the virtual environment.

The framework RV-MTV was presented in previous work [1]. It uses a strategy based on streaming video to allow devices with low computing power — smartphones, tablets or Set-Top Boxes (STB) [1] of Interactive Digital TV (iDTV) and Internet Protocol TV (IPTV) — to interact with applications running remotely. The applications run on high-performance rendering servers. Audiovisual outputs produced by the application are captured and encoded using audio and video compression techniques. They are sent on the fly to client's device via streaming through IP network, or air or cable broadcast. Using the RV-MTV one can build applications that allow viewers from environments with restricted processing power, as iDTV and mobile environments, to interact with HPM objects executed remotely. Controlling the HPM object requires writing an application to send messages according to a specific protocol to the HPM object. Despite the possibility of interacting with an HPM object, it would not be possible with this model to integrate HPM objects into a multimedia presentation. An independent video, subject only to the mechanisms implemented specifically for its control and not related to other media would represent an HPM object.

This paper presents a transparent solution to incorporate HPM objects into multimedia presentations. A dedicated player module for HPM objects is proposed. This player performs the interface between the multimedia presentation machine and the HPM object, translating instructions received from

Computer Science Department - LINCE/UFSCar [1]
Computer Science Department - LaVIIC/UFSCar [2]

[1]The STB term is used in the text to refer to both the set-top box for iDTV and to TVs which have embedded the features of a set-top box

the presentation machine to commands for the HPM object. It is also proposed an approach for mapping semantic concepts common to declarative synchronization languages (anchors, links) to HPM objects. As a proof of concept, we extended the WebNCL [2], a presentation machine for Nested Context Language (NCL) [3], to add support for HPM objects. A multimedia presentation, integrating an HPM object generated by a graphic cluster, was used for testing purposes. Applications considered for such a proof of concept, unlike high-end video games, do not require high responsiveness. Adding to this the image quality acceptable for such applications becomes feasible to use efficient compression techniques and the internet as a distribution platform. A deeper description of the application developed is presented in section 4.

The remainder of this paper is organized as follows: section 2 presents some related works; section 3 defines the HPM object and discusses different strategies to enable them to run on a IDTV environment; section 4 discusses approaches to run remotely HPM applications; section 5 presents the proposed mechanisms to aggregate HPM object to multimedia presentations; section 6 describes a proof of concept developed and section 7 presents conclusions and future works.

II. Related Work

Cesar, Jansen and Bulterman proposed in [4] a way to enrich the multimedia content broadcasted in the context of IDTV. The proposal is to allow viewers to add their comments to the content transmitted by the broadcaster. The paper also discusses conceptually the incorporation of this facility in a declarative language, however it does not present or make any reference to a enrichment by incorporating an interactive media/animation in the IDTV declarative environment or a proof of concept of the ideas.

A possible approach to integrate HPM objects to multimedia presentations in the context of DTV, adopted in some works as in Dias et al [5] and Azevedo and Soares [6], is to rely on local high performance computation by extending the IDTV middleware or part of it (for example, a custom player) for supporting 3D rendering. This approach may be a solution when the hardware of STB, usually quite limited, evolves to larger processing power, memory and graphics capability. That could lead to an increase in the cost of such devices and the need to rewrite applications for the languages supported by the embedded execution engines, which may not always be possible because of third-party resource dependencies, such as libraries and frameworks.

The strategy of remote execution of applications based on HPM objects is also considered in the work of Jurgelionis et al [7]. In that paper the authors propose strategies for implementation, capture, transmission and remote control of games, allowing computers with low computational power to interact with applications. In another related work [1], the authors extend this concept to IDTV and mobile devices and present a framework that enables one to interact with remote applications via a strategy of video streaming. In both works, applications are not integrated into the context of multimedia presentations. They are remotely controlled by commands built with imperative languages.

In the paper of Schmitz [8], the objective is also to enhance multimedia presentations defined by a declarative language. The approach proposes an extension of the language to manage hybrid objects such as animation and video Thus, it combines the control of a video sequence with the flexibility of an animation, highlighting the importance of synchronization of user interactions in a multimedia environment. The solution is restricted to the web and, as the generation of the hybrid media, respecting user interaction, is performed locally. Collaborative interventions between remote users over a shared media are not supported.

Rahman, Hossain and Lornav [9] propose the inclusion of a 3D media in multimedia objects using a declarative language. However, the 3D media is treated standalone. The authors do not address its integration with other media on the composition of a multimedia object.

III. HPM and iDTV

The class of HPM objects defined in the context of multimedia systems refers to interactive applications that produce results perceived to the human senses, usually sight and/or hearing, of very high quality and realism. The results arise from reaction to external stimulus caused by a human interaction, by natural events or events generated by other applications. The resultant media are consumed by humans and must be created with time constraints (short delay between stimulus and response) in order to keep the quality of experience (soft real-time).

Due to its characteristics, HPM objects require high performance computers or clusters to run. Thus, environments with restricted processing and/or memory, such as smartphones, STB and tablets are not suitable to run HPM objects.

Two approaches may be adopted to present HPM objects in IDTV environments. The first one is the inclusion of high performance graphics chipsets in STB, aiming the local processing of HPM objects. The second approach is to run HPM objects remotely on high performance clusters, and then transmit the visual and audible results to the STB.

The approach of local processing may prevent the reuse of legacy HPM objects, developed for running on clusters or high performance computers — environments very flexible as development platforms — due to constraints like differences on the programming languages supported. It would not be possible, for example, the reuse of HPM objects written in C / C + + in an environment that only understand iDTV applications written in Java. Moreover, this approach would not scale and would turn obsolete the chipsets when new types of HPM objects were designed requiring more processing power than the available one.

With the approach of remote processing using clusters, the restriction to reuse legacy HPM objects no longer exists. In addition, it becomes easier to build collaborative applications, since the execution of HPM objects from different users can be held in the same cluster. However, this approach requires that visual and / or audible results from an HPM object execution be transmitted via data network to the STB, which can increase the delay in consumption of the results.

Considering the remote processing approach for HPM objects that returns high quality visual experience to users, one can use different ways for transmitting the output produced to the STB: (1) transmitting uncompressed frames, (2) transmission of pre-rendered frames and (3) streaming of compressed digital video.

With approach 1, the latency time would be low as there would not be delay due to the compression and decompression processes. However, transmitting high quality uncompressed frames requires a large bandwidth, which is not compatible with the reality of the Internet today or terrestrial broadcasting systems. Huang et al. [10], proposed an open cloud gaming called GamingAnywhere, which, according to the authors, is the first open cloud gaming system.

When using pre-rendered frames (2), there would be a negligible increase in latency and a significant reduction in the bandwidth needed to transmit the high quality visual information back to the viewer. However, this approach assumes that local machines have enough processing capabilities to complete the rendering process, which cannot be true with many STB and mobile devices.

Streaming compressed digital video (3) allows the transmission of high quality video over a reasonable bandwidth network. STB and mobile devices usually already have hardware support to perform video decompression. However, the problem with this approach is the increasing in latency due to the time spent in processes of compressing and decompressing video.

There are studies in the literature that seek solutions to the latency of video encoding processes. In the Holub et. al. [11] high-definition videos, which are more costly to be encoded, are considered. It is also important to consider the increasing in response time introduced by running applications remotely, which may be critical for some classes of HPM objects. Barker and Shenoy [12] explore this issue through empirical analysis of the performance of collaborative applications running on cloud computing.

In this work it was adopted the approach of processing HPM objects remotely in clusters and transmitting the results via streaming compressed digital video. The framework RVMTV, which is able to perform the transmission of objects with latencies as low as one second [1], was used in previous works for capturing the HPM object video output, encoding and transmitting the video. However, in this paper, was implemented a more extensible solution for capturing, processing and streaming the video data. While applications like action electronic games, that require instant feedback, fit the definition of HPM, the proof of concept implemented in this work obtains better results with classes of HPM objects more tolerant to greater response times, like virtual tours in museums and those presented as example in Section 1.

IV. Delivering HPM objects processed remotely

In this section, we detail the challenges that are needed to be overcome to enable the delivering of HPM objects processed remotely.

A. Remote Real-time rendering

Real-Time rendering refers to the (necessary low) delay a graphical application takes to render a new visual output in response to a stimulus that changed its state. Usually, graphical applications that are rich in details and with high-quality textures present a bigger delay, which can impair the user experience. However, an application with less visual quality also affects the user experience.

In the case of remote processing a HPM object, the visual quality also affects the bandwidth necessary to transfers the application's output to the users.

B. Graphic Rendering Capture

The capture of graphical rendering is a technique used in several applications: (i) software for capture presentations (especially when animations are used); (ii) recording the user's interaction with a system for the confection of a video tutorial; (iii) capture of a videogame's output for enabling remote play or streaming via Internet, among others.

The approach used for capturing depends on the technology used for rendering. It can be implemented in software or by means of specialized hardware — the second approach leads to a solution independent from the rendering technology used.

An important characteristic of a graphical rendering technique is its capture latency, which is the delay from the instant the rendering engine finalizes a frame to the instant it is copied (captured). It can be measured by comparing the number of frames per second (fps) of the rendering engine produces with the number of fps captured.

Another characteristic is the impact of a capture technique on the graphical application or system performance. It can be measured by comparing the number of fps the rendering engine produces without the capture with the number of fps produced when the capture technique is running. There are four main rendering capture techniques: (i) via window manager API, (ii) via 2D/3D graphic acceleration API, (iii) via mirror drivers and (iv) via hardware component.

- **Window Manager API:** this technique is based on the redraw operations of Window Manager (2D), such as GDI for Microsoft Windows and X11 for GNU/Linux based systems;
- **2D/3D Graphic Acceleration API:** it performs the capture directly from the rendering pipeline from the 2D or 3D graphical acceleration API such as Microsoft DirectX or OpenGL;
- **Mirror Drivers :** this technique is very used in Virtual Networking Computing (VNC) clients to enables remote interacting with a system. However, since the mirror driver is not the default driver for visual output (which is the driver connected to the display or GPU), there is an additional delay between the rendering and the copy to the mirror driver;
- **Hardware Component :** this solution uses a hardware component which is physically connected to the GPU, capturing the output like a PC's display. The captured frames are then compressed and packaged to be delivery

via a data network. Although this solution presents the lower latency, it requires the use a specialized hardware.

C. Real-Time Pixel Streaming

The need for accessing audiovisual content without completely download it stimulated the creation of video streaming services, as known as Pixel Streaming. Roughly, the services of streaming can be classified in Video on Demand (VoD) and Live Streaming.

The video streaming process is composed of the stages: encoding, streaming and playback. The encoding refers to the encoders used to compress video and audio, as well as the multiplexing both stream in consistent media. The streaming is when the data is formatted and segmented in packages, according to a streaming protocol. Finally, the playback consists of grouping back the received packages, demultiplexing the audio and video back, decoding the streams and present then to the user.

Usually, the step that demands more resources, including processing time, is the video encoding. This step presents a direct relation with the video latency, the time delay between the instant in which the original image is generated and the instant in which the coding process is finished. With a higher latency is possible to obtain a better compression, because it is possible a higher reduction on temporal redundancy among the frames. Therefore, it is challenge to specify the encoding parameters for a real-time pixel streaming because it is necessary to harmonize the requirements for low bandwidth, low latency and high visual quality. Another technical challenge is the target playback devices compatibility with the encoding and streaming protocol. Some devices are only compatible with a certain combinations of encoding and streaming protocol, and this may influencing the design of flexible systems that aims to different devices.

D. Remote Interaction

In order to enable the interaction with the remote HPM object, interactions performed on the local device need to be captured. These local interactions also need to be mapped onto interactions that exist in the remote application. For instance, from a mobile device such as smartphone can be mapped the touchscreen interaction onto a mouse click in the remote application. Interactions captures need to be translated into a message following a common protocol known by the remote application. This message is then sent to the remote application by a reliable channel. This communication protocol can be extended to support in addition to user interaction events, triggering of an internal function. When the remote application receives the message, it translates it to the corresponding interaction. The interaction can be inserted into the application simulating a common system event. For instance, the mouse's movement simulation. Otherwise, the interaction can be applied directly to the application, when the HPM object supports it.

The interaction system used in RV-MTV was developed out-of-box and did not consider the presentation machine. Each application should map its interaction into events via the RV-MTV interaction API. Although this approach is still possible in our current implementation, it also complies with the interaction supported by the presentation machine. An HPM object integrated to a multimedia presentation, which uses only the interaction supported by the presentation machine, does not need to handle the interactions by itself — they are responsibility of the presentation machine.

V. HPM as a Multimedia Presentation Component

Multimedia presentations, which define the behavior of multimedia presentations and interactions, can be produced using imperative languages such as Java or Lua, declarative languages, such as SMIL and NCL, or combinations of both, in which imperative scripting languages provide support to the declarative one, as NCL and Lua in the Ginga middleware [3] or HTML and JavaScript.

The combination, declarative language supported by imperative scripting language, is a good compromise since, at the same time it eases the specification of synchronism between media it also provides facilities to define specific situations that may not be adequately supported by the declarative language, but may be implemented with the scripting language.

For the integration of HPM objects to multimedia presentations, this paper explores the possibilities of a declarative environment with support for imperative language. Two alternatives may be considered. The first, referenced here as *Imperative*, is based on API in the imperative language that offers facilities for HPM objects manipulation, such as synchronization, communication and delay control. The second approach, referenced as *Declarative*, as the declarative language resources are extensively exploited, defines direct interactions between the object and the HPM presentation machine. In this case, the HPM objects are in accord with the rest of the media presentation. The advantages and disadvantages of each alternative are presented below.

A. Declarative vs. Imperative Approach

One advantage of the Imperative approach is the flexibility for future changes and specifications due to the ease of modifying the developed API to add new features. Another advantage of this approach is a better decoupling of the API implementation and the presentation machine, which do not need to be modified. Furthermore, an imperative language always gives greater freedom for software development.

However, the Imperative approach creates a distance between the presentation machine and the HPM objects. This gap hinders the integration and interaction of HPM objects with other supported media, because the management of the HPM object is no longer under control of the presentation machine itself. Furthermore, the development of multimedia presentations using the Imperative approach is more complex, as it requires greater skill in programming.

In contrast with the imperative approach, declarative implementation uses the concept of presentation integrated to media management, leading to a more cohesive and natural manipulation of the multimedia document. In this approach the

HPM can be used as a conventional media (such as video or audio), with support to specific demands of HPM objects, such as the address of the remote machine and the communication channel configuration. Synchronization between HPM objects and other media from multimedia object with this approach is straightforward.

The integration of HPM objects to multimedia presentations is more coupled to the presentation machine in the declarative than in the imperative approach. Specifications of the HPM objects behavior can be carried out using the same semantics of the declarative language applied to other media, making use, for example, of interaction concepts common to multimedia declarative languages, such as anchors, regions, etc. In the imperative approach, delegating the handling of HPM objects to an external controller would require re-implementation of these concepts in a language that was not designed for this.

Considering a declarative language such as NCL, for example, which allow transparent addition of new types of media, one can add support for HPM objects without the need to change the language grammar [13] . As a result of the delegation of responsibility for interpretation of the new elements to the presentation machine, this approach strengthens the link between multimedia presentations (with HPM objects) and the presentation machine, since the presentation machine must be extended to support HPM objects.

B. Integration Proposal

Due to the advantages of the Declarative approach presented before, that was the way we adopted for the integration of HPM objects to multimedia presentations, promoting a development model more suitable for multimedia contexts. It is necessary that the declarative language be media-agnostic, which means be able of handling and synchronizing transparently different media, without knowing the media. Examples of languages that presents such characteristic are NCL and SMIL [14].

This class of languages uses transitions in the state machine of each media to manage synchronization. For example, synchronization points may be the beginning, end or pause of a specific media. Another intrinsic characteristic of these languages are the concepts of media properties and media anchors. The anchors refer to portions of media, such as a time interval in a video. The properties are features that can be accessed and changed during the presentation, for example, the audio volume.

To promote the incorporation of HPM objects in multimedia presentations, some changes must be done in the presentation machine to support this new class of objects. It is necessary to map the concepts of properties, anchors and state transitions onto HPM objects. The interactions between the presentation machine and the HPM object must be encapsulated in control messages exchanged via a communication channel. Figure 1 illustrates how the concepts of anchor are mapped onto HPM objects and how HPM objects can be integrated into multimedia presentations. A syntax similar to NCL is use to exemplify the integration.

There are two regular media (video1, paint1) and a HPM object which is been produced by a remote application. When

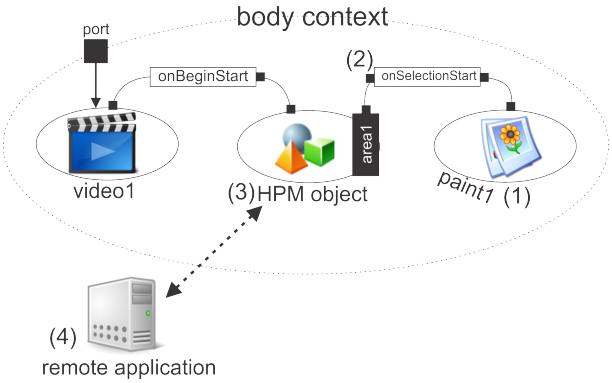

Figure 1. HPM Object in a multimedia presentation

the presentation starts, the video1 starts to play because it has a port. Moreover, a link defines a casual relation between video1 and the HPM Object. This relation defines that when the video1 begins it execution, the HPM object most start.

HPM object also has a anchor called area1 (item 2), and this anchor has a causal relation defined by a link with the media paint1 (item 1). When paint1 is selected (i.e. the user click on it), the area1 anchor must start. When the HPM player (item 3) is notified that it should start the area1, it sends a message to the remote application (item 4). The remote application will apply the result of the event in the virtual environment (in this case modifying the current position of the user in the virtual environment of to the position specified by the anchor), thereby updating the produced video streaming. When the HPM player sends an event to the remote application, it also pauses the video streaming until it receives an acknowledgment message from the remote application, which means that the event was processed.

Synchronization: Due to the delay inherent to the process of video encoding, decoding and transmission, the video presented to the viewer does not reflect the current state of the application running on the remote server. Although the synchronization between the states is not critical as HPM objects require only a sufficient synchronization to human perception, the latency in response time can affect the quality of experience. The presentation machine extension to integrate HPM objects must provide mechanisms to circumvent this problem.

In order to reduce the problem of states inconsistency, a synchronization mechanism between the state of the presentation and the state of the HPM object, using the clock of the involved machines (timestamp), is proposed. An initial calibration is required to synchronize the clock of local and remote machines. The calibration is repeated periodically to guarantee the sync. This synchronization does not need to be exact, tolerating differences that are not relevant to humans' senses.

The application keeps, for each rendering iteration, the current state of its objects (e.g. user three-dimensional position and point of view). The presentation machine contributes with the sync process adding a timestamp in the messages when

mapping events to messages and stop accepting new event requests until get confirmation message from the remote HPM object.

Further, to counterbalance the video coding/decoding delay, it is necessary that the remote application be able to evaluate the time spent in the process. Adding this time with the difference between the timestamps of control messages, the application can return to the state the presentation was when it sent the event.

There is also the need to address possible remote application freezes, discovered by the lack of response message. To do that, the extended machine triggers a timer waiting for a confirmation. In the event of a timeout, the presentation machine takes steps to alert the user, to reestablish communication or to replace the remote server.

Application Manifest: To make simple the integration of HPM objects into multimedia presentations, we use an XML file to specify the characteristic of the HPM object. This file can hold information about the video stream address generated from the HPM object and the ways by which messages can be sent to the remote application that is generating the HPM object. It also lists the proprieties and anchors of the HPM object by which the multimedia presentation can interact with the HPM object. The HPM object's authors should provide this Manifest File.

The Listing 1 contains a manifest file sample. In the lines 1 and 2 is defined the address of the video stream produced by the remote application. Between lines 3 and 8 the HPM object anchors and propriety are listed. Between lines 9 and 13 is specified how the communication with the remote application will be performed — in this case, via a message broker. Between lines 14 and 17 is specified the alternative content that will be displayed by the latency dissimulating mechanism.

```
<manifest class="HPM">
  <media type="video"
    mrl="rtmp://mediaserver/streaming" />
  <anchors>
    <area name="PAINT_001" />
    <area name="PAINT_002" />
    <area name="PAINT_003" />
    <property name="speed" />
  </anchors>
  <communication protocol="stomp">
    <address brokerURI=
   "failover:(tcp://broker:61616)" />
    <topic dest="HPM_APP" />
  </communication>
  <holdOn>
    <waitingTrick type="image"
      mrl="loading.gif" />
  </holdOn>
</manifest>
```

Listing 1. NCL Link

VI. Proof-of-Concept

Our proof of concept for the proposed integration of HPM objects into multimedia objects is composed of two parts. The first part is the extension of an NCL presentation machine to support HPM objects as a new media type, which was done by adding a new media player for HPM objects. The second part is building a distributed application, consisting of an application running on high-performance clusters that produces HPM object and a NCL document, which embedded this HPM object. The virtual environment used was the *Torre de Papel* by Sonia Menna Barreto [15]. As the environment of the validation and testing of our proof of concept we used a local network composed by the server, a machine capable of running smoothly the 3D application (60 frames per second or more), on a wired connection bandwidth 100Mbps and with a latency to the router of 1ms or less. There are two types of customer, a laptop and a tablet, both with a wireless (802.11n) and a latency lower than 10ms between customer and the server. For the approximation of the actual conditions of use of the solution on the Internet the latency between the end points was increased by 80ms, which is an acceptable latency in the region of Brazil [2] For network monitoring, bandwidth and latency analysis, we used a tool for network sniffer [3].

A. HPM Remote Solution in TVDi

In the diagram showed in Figure 2 are observed modules components of remote interaction solution with HPM, as well as its two main features: (i) sending the rendering of HPM object captured in the form of a pixel stream (upper arrow) and the receiving of the remote user interactions (bottom arrow).

It is also shown systematic progress that is used to perform the pixel streaming. The first step is the insertion of dynamics library into the HPM object (clearly higher arrow). After that, the capture module begins receiving rendering bits captured from HPM object and arranges data as an image (bitmap) and sending it to the encoding module. This module encodes the images, using a selected video encoding, multiplexes streams, and send forwards it to the streaming module. The stream module packages the multiplexed stream and streaming it as a pixel stream.

Finally, the pixel stream is received and played by the playback module. At the bottom of Figure 2 are showed steps to perform remote interaction with the HPM object. First, the control module intercepts the user interactions (mouse, keyboard, touchscreen, etc.). After that, it sends via an established connection (using a message broker) these commands to the interaction module, using a common message protocol. This means of communication eases and decouples the interaction module and the control module. In conclusion, the interaction module maps these events into interactions directly in the HPM object.

For the implementation of the prototype solution for HPM remote interaction were developed the following modules:

- Capture Module: responsible for all operations related to screen capture and HPM audio application;
- Encoding Module: responsible for preparing capture frames of images and encoding them in a chosen standard;

[2] SIMET - http://www.ceptro.br/CEPTRO/SiMeT

[3] Wireshark - http://www.wireshark.org/

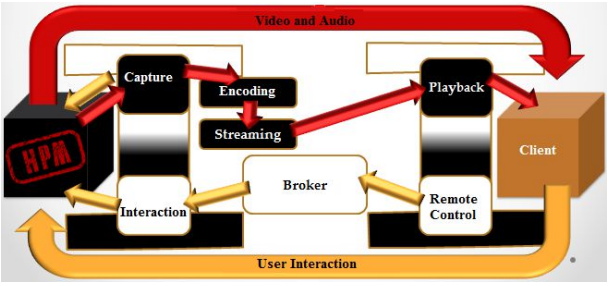

Figure 2. Components Diagrams of HPM Interaction Solution

- Playback Module: responsible for playing the video streaming received at the web page;
- Control Module: it intercepts all user interactions with the web page and sending commands to the remote interaction;
- Remote Interaction Module: responsible for mapping commands in HPM interactions.

B. Adding a HPM Player to the presentation machine

The presentation machine we choose to extend was the WebNCL [2], an NCL presentation machine developed on top of web stack technologies (HTML/CSS /JavaScript) that allows the execution of NCL documents on HTML5 compatible browsers.

The WebNCL's Event Manager is responsible for controlling the processing of NCL links. It is media-agnostic, which means it does not need to know if the media is an image or a video — or even non-standard type such as HPM objects — to perform the processing of events. The responsibility to inform the event manager about the transitions occurred in the media presentation that can represent conditions in NCL links — as beginning of an anchor — is delegated to players of each media. In addition, the media players are responsible for receiving and processing the actions triggered by links — as modify a media property.

Because of this decoupling between the event manager and the media players in WebNCL, in order to add support for HPM objects to WebNCL we needed to implement a new media player. This new player is responsible for playing and controlling HPM objects. This approach also does not require modifications in the presentation machine core.

Figure 3 depicts the WebNCL's architecture and details the HPM object player that was added to the presentation machine. The HPM player was inserted into presentation layer, the same layer where the other media players are situated. It is composed of components responsible for parsing the manifest file (Manifest Parser), handling and presenting the video streamed by the remote application (Video Stream Player), exchanging messages with the remote application (Communication Module) and synchronizing and performing dissimulation delay mechanisms (Synchronization Module).

The HPM object player receives video stream using RTMP (Real Time Messaging Protocol)[4] and displays the video

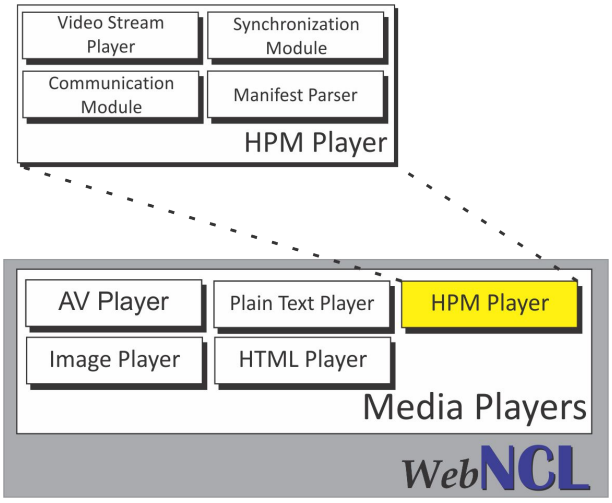

Figure 3. NCLweb and HPM Player Architecture

frames on the screen in the position defined by the presentation machine. Whenever an action event is triggered over an HPM object media node — as to start a certain anchor — the HPM player encodes that event in a JSON (JavaScript Object Notation) [5] message and sends it to the remote application.

In a similar way, when the remote application notices any transition that can be used as a condition to NCL links — such as HPM object's anchors starts — it sends a message to the HPM player. The HPM player decodes the message received and triggers the corresponding events to the presentation machine.

The HPM player also implements synchronization mechanisms to work around possible delay resulting from the video encoding strategy. When a user stimulus can generate some inconsistencies in the presentation due to the difference between what the user see and the HPM object state in the cluster, the HPM player sends a message to the remote application with the video package timestamp. The remote application uses the timestamp to retrieve the HPM object state for the moment when user's stimulus has been happened. In this state, the remote application processes the stimulus, producing the right visual output. While the cluster processes the user's stimulus and adjust the HPM object state, the HPM player can display some alternative content to dissimulate latency.

C. HPM object embedded in a NCL presentation

Figure 4 depicts the infrastructure used for integrating HPM objects into an NCL presentation.

The graphic cluster is an instantiation of a Cave Automatic Virtual Environment (CAVE) [16] called Mini CAVE [17]. The graphics rendering applications is done in a distributed way. Thus, complex environments can be presented with a high frame rate.

The running application was developed in C++ using the

[4]RTMP - http://www.adobe.com/devnet/rtmp.html

[5]JSON - http://www.json.org

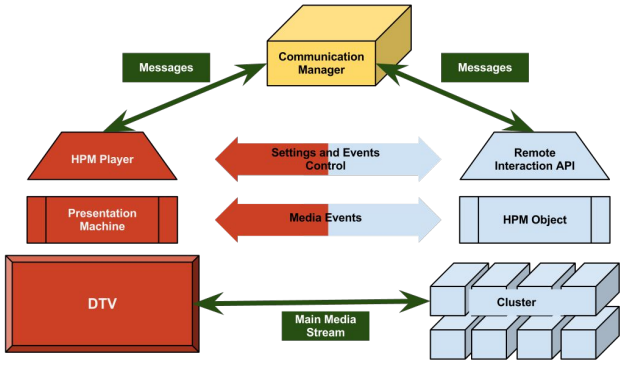

Figure 4. HPM Architecture

OGRE 3D [6], a graphic engine. It runs on a cluster of six nodes, equipped with Intel i7 processors and NVIDIA GPU.

The application responsible for generating the HPM object is an 3D modeling of a painting of Sonia Menna Barreto, a Brazilian artist, called *Torre de Papel*. The environment has some predefined animations and other paintings scattered by the environment, such as an art exhibition of Sonia's works. Users can navigate in the virtual environment using traditional devices (mouse, keyboard) and unconventional ones (mobile devices, Kinect and Wii Remote).

The application has been modified to incorporate the components for performing pixel streaming. Using the components, the rendering frames are captured and encoded, generating a video stream that is sent to the HPM player.

The anchors of the HPM object were defined as certain positions within the 3D environment (the coordinates x, y, z and the camera angle by which the image is seen). Each anchor points to an environment location that faces one of the exhibited paintings. When the application receives a message from the HPM player informing a *play event* in a certain anchor, the application adjusts its display to show the painting associated with that anchor. One can also set the predefined animations speed by modifying the value of speed property via events sent by HPM player.

In addition, it was implemented a new interaction mechanism, allowing navigation via NCL application. If a user, navigating in the environment, approaches to a painting associated to an anchor, the application will send an event to the HPM player, informing the start of an anchor. When the user leaves the area, an event indicating the end of an anchor will also be sent in an analogous manner. Figure 5 shows the application running on the cluster graphic.

A module for controlling the messages exchanged between the HPM Player and the HPM object running on the cluster is important. We chose to use the ActiveMQ broker [7] to accomplish the management of these messages.

[6]OGRE 3D - http://www.ogre3d.org/

[7]ActiveMQ - http://activemq.apache.org

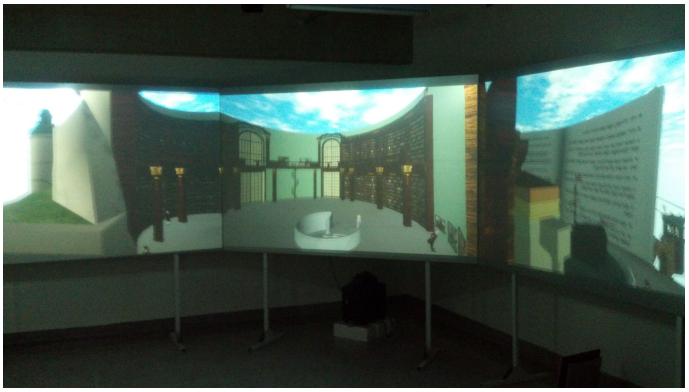

Figure 5. HPM object running on the cluster graphic

The advantages of a using a broker for message exchange are its reliability and security. Furthermore, as libraries for integration with ActiveMQ are present in many different languages, the construction of distributed applications on different platforms and languages, as occurs in this proof of concept, is facilitated.

We used the STOMP protocol (Simple Text Message Oriented Protocol) [8] to realize the communication between applications, encoding messages in JSON format. Streaming video produced by the application is sent to a remote media server. The media server then retransmits the video for the presentation machine using a multimedia streaming protocol (RTMP).

The NCL document, which embed the HPM object, allows users to navigate among the anchors through a menu. When the user is viewing any HPM anchor, a high-resolution image of the associed artwork is displayed. The user can also change the speed of the animations in HPM object using the colored buttons in the remote control.

```
<media id="interactivePaint"
       src="HPM_manifest.xml"
    descriptor="dInteractivePaint">
  <area id="paint1" text="PAINT_001" />
  <area id="paint2" text="PAINT_002" />
  <area id="paint3" text="PAINT_003" />
  <property name="speed" value="1.0" />
</media>

<link id="lGoPaint1"
    xconnector="onSelectionStopStart">
  <bind component="menuItem1"
    role="onSelection" />
  <bind component="interactivePaint"
    role="stop" />
  <bind component="interactivePaint"
    interface="paint1" role="start" />
</link>

<link id="lRedKey"
    xconnector="onKeySelecionSet">
  <bind component="interactivePaint"
    role="onSelection">
        <bindParam name="keyCode"
```

[8]STOMP - http://stomp.github.com

```
    value="RED" />
  </bind>
  <bind component="interactivePaint"
    interface="speed" role="set" >
        <bindParam name="setValue"
    value="1.5" />
  </bind>
</link>

<link id="lShowPaint2"
    xconnector="onBeginStart" >
  <bind component="interactivePaint"
    interface="paint2" role="onBegin" />
  <bind component="plainPaint2"
    role="start" />
</link>

<link id="lHidePaint2"
    xconnector="onEndStop" >
  <bind component="interactivePaint"
    interface="paint2" role="onEnd" />
  <bind component="plainPaint2"
    role="stop" />
</link>
```

Listing 2. NCL Link

Listing 2 presents some excerpts from the NCL document. Between lines 1 and 8 a HPM media node is defined. The media's source is the HPM manifest (HPM_manifest.xml). It describes the possible properties, anchors and how to communicate with the HPM object (the address of streaming and other information that defines the object). Inside the media node are defined three anchors and one property.

The link lGoPaint1 (lines 10-18) defines an action that causes the HPM object to be moved to the anchor Paint1 when a particular menu icon is selected. The link lRedKey modifies the value of HPM oject speed property to 1.5 when the red button is pressed, in other words, increasing the speed of the animations in 50 %.

The links lShowPaint2 (lines 34-40) and lHidePaint2 (lines 42-48) make the image plainPaint2, a high-resolution image of an artwork, be presented when the user is near to the equivalent painting in the HPM object.

Figure 6 shows the NCL document being presented. The user can interact with the application using the arrow keys to navigate between artworks from a side menu. When the user selects a menu item, the HPM object is moved to its anchor and a high resolution image of the artwork is displayed (Figure 7).

Figure 6. Navigation in the virtual environment

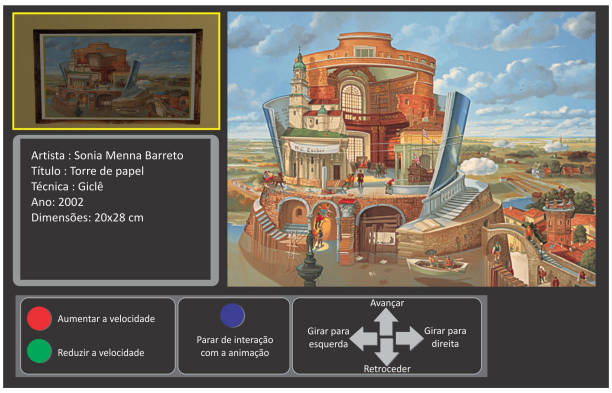

Figure 7. HPM synchronization with other media

VII. Final Remarks

The work presented in this paper confirms the feasibility of the proposed approach for embedding HPM objects in multimedia presentations with a platform-decoupled solution for remote processing HPM applications. From a multimedia presentation platform with declarative language, complemented by imperative scripting language, interpretation capabilities, the approach adopted was to promote changes in the presentation machine so that the new media could be fully managed by the declarative part of the environment only. Thus, HPM objects can be better integrated with the multimedia presentation, as all its management is kept under the control of the presentation machine. It was also shown the feasibility of incorporating HPM objects on TVDi and mobile devices environments, using a strategy of video streaming.

It was presented as proof of concept the upgrade of a NCL presentation machine needed to enable it to run multimedia presentations that have HPM objects as one of their media. The presentation machine manages the new media generically, as it was an image or a video. Upgrading the presentation machine demanded no modification in the grammar of NCL as it is a media-agnostic language. A NCL multimedia application, including an HPM object generated remotely by a graphics cluster, was also developed and tested. This experience showed that creating a multimedia presentation with HPM objects is natural for developers who already have experience in authoring NCL documents. Also, it was developed a modularized solution to allow the remote processing and control of a HPM application with minimal impact in such processing and with low latency in video capture and streaming. The solution reached less than 4 Mbps for a HD live stream and increasing only 27ms (or less) in latency between the production of the media and the client consumption.

The main issues that may be addressed in future works, in order to improve the proposed approach of embedding HPM objects in multimedia applications for IDTV and mobile devices environments, are related to the delay of encoding and

streaming video. One can explore an approach based on the generation of pre-rendered frames at the cluster, leaving the final rendering to the local application. This process can be improved with embedded GPUs as OpenGL-ES [9] or even a web version like WebGL. Another future work could be the study of different client technologies to the remote control module for a more extensible compatibility with different OS devices.

ACKNOWLEDGMENTS

The authors acknowledge the financial support provided by CAPES and CNPq.

REFERENCES

[1] C. C. Viel, E. L. Melo, L. Trevelin, and C. A. C. Teixeira, "Rv-mtv: Framework para interação multimodal com aplicações de realidade virtual em tv digital e dispositivos móveis," in *WebMedia 2011*, 2011.

[2] E. L. Melo, C. C. Viel, and C. A. C. Teixeira, "Webncl: A web-based presentation machine for multimedia documents," in *WebMedia 2012*, 2012.

[3] L. Soares, R. Rodrigues, and M. Moreno, "Ginga-ncl: the declarative environment of the brazilian digital tv system," *Journal of the Brazilian Computer Society*, vol. 12, no. 4, pp. 37–46, 2007.

[4] P. Cesar, D. Bulterman, and A. Jansen, "An architecture for end-user tv content enrichment," *Journal of Virtual Reality and Broadcasting*, vol. 3, no. 9, p. 14, 2006.

[5] D. F. Souza, T. A. Tavares, L. S. Machado, and G. L. Souza Filho, "Incorporating 3d technologies to the brazilian dtv standard: a study of integration strategies based on middleware ginga," in *Proceedings of the 8th international interactive conference on Interactive TV&Video*, ser. EuroITV '10. New York, NY, USA: ACM, 2010, pp. 251–258. [Online]. Available: http://doi.acm.org/10.1145/1809777.1809828

[6] R. G. D. A. Azevedo and L. F. G. Soares, "Embedding 3d objects into ncl multimedia presentations," in *Proceedings of the 17th International Conference on 3D Web Technology*, ser. Web3D '12. New York, NY, USA: ACM, 2012, pp. 143–151. [Online]. Available: http://doi.acm.org/10.1145/2338714.2338739

[7] A. Jurgelionis, P. Fechteler, P. Eisert, F. Bellotti, H. David, J. P. Laulajainen, R. Carmichael, V. Poulopoulos, A. Laikari, P. Perälä, A. De Gloria, and C. Bouras, "Platform for distributed 3d gaming," *Int. J. Comput. Games Technol.*, vol. 2009, pp. 1:1–1:15, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1155/2009/231863

[8] P. Schmitz, "Multimedia meets computer graphics in smil2.0: a time model for the web," in *Proceedings of the 11th international conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002, pp. 45–53. [Online]. Available: http://doi.acm.org/10.1145/511446.511453

[9] M. Rahman, M. Hossain, and A. Saddik, "Lornav: A demo of a virtual reality tool for navigation and authoring of learning object repositories," in *Distributed Simulation and Real-Time Applications, 2004. DS-RT 2004. Eighth IEEE International Symposium on*, oct. 2004, pp. 240 – 243.

[10] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: An open cloud gaming system," in *Proceedings of ACM Multimedia Systems 2013*, Feb 2013.

[11] P. Holub, L. Matyska, M. Liška, L. Hejtmánek, J. Denemark, T. Rebok, A. Hutanu, R. Paruchuri, J. Radil, and E. Hladká, "High-definition multimedia for multiparty low-latency interactive communication," *Future Generation Computer Systems*, vol. 22, no. 8, pp. 856 – 861, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X06000380

[12] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, ser. MMSys '10. New York, NY, USA: ACM, 2010, pp. 35–46. [Online]. Available: http://doi.acm.org/10.1145/1730836.1730842

[13] L. F. G. Soares, M. F. Moreno, and F. Sant'Anna, "Relating declarative hypermedia objects and imperative objects through the ncl glue language," in *Proceedings of the 9th ACM symposium on Document engineering*, ser. DocEng '09. New York, NY, USA: ACM, 2009, pp. 222–230. [Online]. Available: http://doi.acm.org/10.1145/1600193.1600243

[14] SMIL3. (2012, Maio) êsSynchronized multimedia integration language (smil 3.0). http://www.w3.org/TR/SMIL3/. [Online]. Available: http://www.w3.org/TR/SMIL3/

[15] B. B. Gnecco, D. R. C. Dias, and M. P. a. Guimaraes, "Traditional paintings and the digital medium," *SBC Journal on 3D Interactive Systems*, vol. 2, no. 3, p. 6, 2012.

[16] L. P. Soares, "Um ambiente de multiprojeção totalmente imersivo baseado em aglomerados de computadores," Ph.D. dissertation, USP, São Paulo, 2005.

[17] D. R. C. Dias, J. R. F. Brega, M. Paiva Guimarães, F. Modesto, B. B. Gnecco, and J. R. P. Lauris, "3d semantic models for dental education," in *ENTERprise Information Systems*, ser. Communications in Computer and Information Science, M. M. Cruz-Cunha, J. a. Varajão, P. Powell, and R. Martinho, Eds. Springer Berlin Heidelberg, 2011, vol. 221, pp. 89–96.

[9] OpenGL-ES - http://www.khronos.org/registry/gles/