Abstract
The rapid prototyping, modeling and authoring of virtual environments has been a major issue presented in the literature. However, due to the complexity for describing these environments and the dynamic nature of virtual reality applications it is fundamental to provide means for customizing them according to the interactions of the user and the purpose of the application. The customization allows the modification of the virtual environment without much coding effort and with a greater flexibility. This paper presents two approaches used for customization of virtual environments. By the utilization of XML these approaches provide a flexible and easy way to modify or configure virtual environments. The first one proposes a method to allow the scenario modification for Java3D API environments. The second approach demonstrates how to customize virtual environments with multimedia content using OGRE.

1. Introduction
Virtual Reality (VR) systems have become a relevant solution which has been applied in several domains, such as education, entertainment, heritage exhibition, training, etc. Nevertheless, the development of Virtual Environments (VEs) and 3D applications is a complex task with a high cost which demands advanced modeling and programming skills.

Frequently these VR systems are designed and implemented for a specific area of application. Creating a single, generic authoring tool for every different kind of application would be an impossible task especially if the authors are non-skilled programmers.

A more pragmatic approach would be to consider the creation of generic environments which could be customized by the user according to a particular context. The proposal of solutions to meet this approach would be helpful for novice or advanced users when they need to extend or customize their virtual environments. XML (Extensible Markup Language) can be extensively applied for the customization of VEs. With this language it is possible to both novice and advanced users to create useful virtual reality applications, allowing the smooth migration of novice users to becoming more experienced [1].

VEs can be customized in different ways where users are able to resize the environment, change textures, exclude or include 3D objects, modify interaction behavior, present multimedia content, among others. The customization of VEs seems to be a reasonable solution for providing dynamics to an application. However, this solution can also be a programming demanding task. For this reason, we concentrated our efforts to the proposal of simple and generic (language independent) solutions in order to provide customization without programming needs. This paper presents two of these solutions, illustrating the utilization of XML for providing customization of VEs with 3D objects and synchronized multimedia presentations.

This paper is organized as follows: Section 2 discusses the literature review concerning the
customization of VEs; Section 3 introduces the customization of VEs based on XML and Java 3D API; Section 4 presents the customization of VEs with XML and Ogre-Multimedia, and; Finally, Section 5 discusses some conclusions of this work.

2. Related work

Different approaches have been proposed in the literature to provide users with a flexible and an easy to use interface for building, remodeling and customizing VEs. Among these approaches, Hendrick et al. [2] present a meta-authoring tool as a generic framework for building VEs for both novice and advanced users. Also concerned with the usability of such tools for novice users, Robertson and Good [3] discuss the implementation of an authoring tool which provides the young users with an intuitive interface for customizing 3D story telling narrative based on XML.

Indeed, XML has been frequently applied for the customization of VEs. Mazzoleni et al. [4] also applied XML to provide the customization of VEs with multimedia presentation from digital libraries on virtual guided tours. Boukerche et al. [1] present their solution to provide the extensibility of non-linear 3D stories using XML scripts for dynamically changing its presentation. Rodrigues and Oliveira [5] also introduce ADVICE, an authoring tool based on XML for creating collaborative VEs.

Some other solutions for authoring VEs are also proposed and directed to more skilled users such as XVR [6], developed by VRMedia, which is a development environment dedicated to VR and Augmented Reality (AR) applications based on a C++-like scripting language. XVR is suitable to be deployed both on professional VR installations and on Internet WebPages or multimedia CDROMs. Also, Osawa at al. [7] introduced 3D which is an interactive toolkit library for 3D applications implemented in Java3D API for providing rapid prototyping, portability and customization.

These approaches illustrate some of the different possibilities of customization of VEs. Some of them provide an easy to use environment for novice users, and some others offer a more professional environment for advanced applications. In this paper we present two different efforts for providing the customization of VEs based on XML. These efforts aim at applying an intuitive representation for novice and advanced users for building 3D environments and customizing them with new 3D objects and synchronized multimedia presentations.

3. Configuring Virtual Environments with XML and Java 3D API

The interaction with VEs can be enriched if the user is able to insert, remove or modify some features of the environment and its 3D objects (dimensions and coordinates). These transformations can modify individual features of some objects on the environment, however, in some cases, they can not modify the scenario. The main idea of the approach presented in this section is to provide a framework to allow the scenario modification, using minimal programming efforts.

The solution proposed is based on XML to describe the main features of the scenario and then parsing these features to a basic VE developed with Java 3D API programming language. Figure 1 illustrates the project’s architecture.

![Figure 1. The project architecture](image-url)
Java Classes. The XML files provide the contents, i.e., the attributes that should be used to configure the VE.

Indeed, the main component of the structure is the XML files. They store the data for the configuration. The entire VE can be remodeled or recreated just by modifying the values of the XML files. Two identical XML files are applied: one for static objects and one for dynamic objects.

The configuration of static objects is stored on the file depicted in Figure 2. It stores the attributes to identify and configure the objects individually, such as coordinates, size, appearance (color, transparency, texture) and representation. The static objects are the ones with no interaction. They are only applied for composing the scenario.

The dynamic objects provide some kind of interaction for the user within the VE. These objects can also be associated with multimedia content. These features can be configured using the same file depicted in Figure 2. Each element `<Av>` characterizes an object inside the VE. A description of the tags applied on the XML files is presented on Table 1.

Figures 3 and 4 present illustrations of the VEs generated from the XML files presented previously. Figure 3 presents a VE simulating a virtual room. By modifying the file of Figure 2, the scenario of the room remains the same, but the furniture changes. In reality, this approach maintains the Java 3D scene graph structure.

Figure 4 consists in a free area composed by a street, pavement and lights. This environment was created using few static objects. It is possible to add other objects by changing the content presented on Figure 2.

It is also possible to associate multimedia content exhibition with the objects. The system does not control the activation and deactivation of a multimedia presentation. It works just as a point of interaction to start the respective player. Table 2 presents the media formats supported by the application. They are supported by the native programs of the operating system.
The solution applied for the multimedia support can be observed in Figure 5. This example represents a scenario inside a virtual room. This environment can be applied for educational purposes, for instance. It is important to highlight that the setup of this VE was designed by files where the structure of Figure 2 is customized by the author.

Figure 5(a) presents some furniture where a slide presentation is associated. When the user clicks on it, a power point section is opened. Figure 5(b) shows an image rendered directly on an object. Figure 5(c) illustrates the video support. The tapes on rack represent different video files that can be played by the user. In Figure 5(d) it is possible to observe some books. They are related to text files. The object’s representation of sound media is illustrated on Figure 5(e). The VE uses a stereo channel for executing them. Finally, Figure 5(f) presents all media together.

Table 2. File extensions of Multimedia Contents

<table>
<thead>
<tr>
<th>Media</th>
<th>File extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>TXT, DOC, HTML</td>
</tr>
<tr>
<td>Image</td>
<td>JPG, GIF, BMP, PNG</td>
</tr>
<tr>
<td>Sound</td>
<td>MID, WAV, MP3</td>
</tr>
<tr>
<td>Video</td>
<td>MOV, AVI, MP4, WMV</td>
</tr>
<tr>
<td>Slides presentation</td>
<td>PPT, PPS</td>
</tr>
</tbody>
</table>

Figure 5. Example of virtual room with multimedia
From the structure depicted in this section, it is possible to create a customizable VE through data parsing between XML and Java 3D API objects, without direct code modification of the VE. It allows the VE generation according to the user needs, considering few programming efforts.

4. Integrating Multimedia Content within Virtual Environments

The presentation of synchronized multimedia content can also be customized inside VEs. For this purpose, a solution was proposed and developed for providing the integration of multimedia content within an Ogre's VE, the Ogre-Multimedia.

Ogre (Object-Oriented Graphics Rendering Engine) [9] is a scene-oriented, flexible 3D engine written in C++ programming language designed to make it easier and more intuitive for developers to produce applications using hardware-accelerated 3D graphics.

Due the facility for integrating different APIs with Ogre's library several formats of multimedia content were considered in this approach: the output of any embedded multimedia player (such as RealPlayer [10], GRiNs [11], etc.); a Flash executable content [12], or; a web-browser content. The main goal of Ogre-Multimedia is to provide the customized presentation of multimedia content as textures over any 3D object inside the VE.

4.1. Customizing multimedia presentation inside a VE

Before proposing a customizable solution for the presentation of multimedia contents inside a VE, it was also important to consider important requirements such as the need for specifying synchronization relations among the media objects inside the VE, supporting user interactions with these objects, ensuring interoperability of multimedia players, mapping 2D objects into the 3D world, among others. Unfortunately, the existing languages and models for describing multimedia presentation such as SMIL [13] do not support the description of three-dimensional channels, that is, the specification of the x, y and z coordinates for the presentation of the multimedia content inside the 3D environment. For this reason, the solution relied on the proposal of a simpler XML-based meta-language for describing multimedia documents to be presented inside VEs, or as we called the meta-multimedia document.

The meta-multimedia document was strongly inspired on the syntax of SMIL and can be applied as a multimedia authoring language where users can customize the VE and describe what is going to be presented, where and when they will be presented. Briefly, it describes all the components of the multimedia presentation and their temporal and logical synchronization. The particularity about this presentation is that all the media objects (multimedia documents, flash, web-browsers, and primitive media objects such as video, image, text, audio, etc.) are synchronized and rendered anywhere inside the VE. The interpretation and coordination of this document presentation inside the VE is done by the API developed for Ogre, the Ogre-Multimedia.

As presented in Figure 6, the structure of the meta-multimedia document is composed by four main elements: panel, trigger, eventHandler and event.

Each panel element describes a presentation panel for media objects inside the virtual environment. The container called panels is a set of the panel objects that must be rendered inside a VE.

Each element trigger characterizes an object inside the VE which controls the activation and deactivation of a multimedia presentation. The container called triggers is a set of all the trigger objects that will be used to control the multimedia presentations inside a VE.

Each element EventHandler characterizes how the presentation of the media objects associated with a given trigger will be controlled (e.g., start their presentation when the user clicks on the trigger or when he approximates it). The container called eventHandlers is a set of all multimedia presentation described by all the elements eventHandler.

Each element event characterizes how and when the presentation of each media object of a given eventHandler will be carried out.

The structure of the meta-multimedia document was defined to make the process of authoring the multimedia document easier and intuitive. The meta-multimedia document is considered as the key-solution for the integration of
multimedia content inside vEs. Indeed, with this document, the author of the application is able to customize his ve with new multimedia content without changing a single line of his code.

4.2. OGRE-Multimedia: Integrating Multimedia within Virtual Environments

Multimedia has been applied in different applications as a helpful tool for providing insight about a subject being presented. At the same time, it has been proved that human is more receptive to new information and construct easier cognitive models if this information is presented in different modalities [14]. The integration of multimedia content inside vEs is a promising and interesting trend in the development of vr applications. Indeed, Multimedia captivates user’s attention inside the ve enhancing interaction, promoting user’s interest, facilitating learning and improving his immersion.

Some important issues must be considered for the integration of multimedia content inside a ve such as the specification of the temporal and logical synchronization of different media objects (with at least one audio or video) to be rendered inside the 3D environment, and determining which events (e.g., user interactions) will be applied for the communication between the 2D/3D worlds.

Unfortunately, most of the existing languages for describing 3D environments (such as vrML [15], X3D [16] or Java3D [8]) are monomedia and non-interactive since they support only the presentation of isolated media objects without any synchronization relations among them. One exception to this is MPEG-4 which by means of BIFS allows the creation of rich 2D/3D graphical scenarios with synchronized multimedia [17]. However, the authoring of the MPEG-4 BIFS is still too complex and intuitive tools and approaches are still lacking.

The solution proposed with ogre-Multimedia is to provide an API to integrate the presentation of different multimedia objects (rendered by different plug-ins or APIs) around the definition of the meta-multimedia document. In this sense, the meta-multimedia document describes all the synchronization relations among all the components (media objects, Flash presentation, web-browsers, etc.) of the document. Taking advantage of the ogre’s component-based architecture, this API can be easily instantiated.

Figure 6. Example of a meta-multimedia document

```
<multimediaControl>
  <panel>
    <panel name="MainUMa" width="1024" height="768" scale="0.2" position="745, -150, 0" verRotation="90" />
    <panel name="LeftUMa" width="460" height="480" scale="0.3" position="745, -150, 153" verRotation="90" />
    <panel name="TopLeftUMa" width="460" height="480" scale="0.3" position="700, 15, 153" verRotation="90" hozRotation="45" />
    <panel name="TopUMa" width="460" height="480" scale="0.3" position="700, 15, 0" verRotation="90" hozRotation="45" />
    <panel name="TopRightUMa" width="460" height="480" scale="0.3" position="700, 15, -153" verRotation="90" hozRotation="45" />
    <panel name="RightUMa" width="460" height="480" scale="0.3" position="745, -150, -153" verRotation="90" />
  </panel>

  <triggers>
    <trigger name="TriggerUMa" position="-535, -174, 0" scale="1.5" verRotation="45" />
  </triggers>

  <eventHandlers>
    <eventHandler triggerName="TriggerUMa" action='click' loopEvent="true">
      <event source="MyPlug" volume="25" />
      <event panelName="MainUMa" source="http://www.uma.pt" />
      <event panelName="LeftUMa" source="canvas2.js" start="3s" stop="16s" fadeOut="1s" />
      <event panelName="TopLeftUMa" source="FebadaLin1.js" start="3s" stop="16s" fadeOut="1s" />
      <event panelName="TopUMa" source="Biblioteca.jpg" start="3s" fadeOut="16s" stop="16s" fadeOut="16s" />
      <event panelName="TopRightUMa" source="ENC.js" start="3s" stop="16s" fadeOut="1s" />
      <event panelName="RightUMa" source="Biblioteca.jpg" start="16s" fadeIn="16s" stop="16s" fadeOut="1s" />
      <event panelName="RightUMa" source="Anditecrol.js" start="16s" fadeIn="16s" stop="16s" />
    </eventHandler>
  </eventHandlers>
</multimediaControl>
```
and integrated with the remaining available library’s API. This section presents the main architecture of OGRE-Multimedia.

The architecture of OGRE-Multimedia describes the integration of the meta-multimedia document with the VE, which is supported by the implemented software modules and some existing APIs. This architecture is depicted in Figure 7.

The architecture of OGRE-Multimedia (Figure 7) is composed by four main components:

(i) External modules, which are represented by those APIs developed by other projects, or which were already provided by the OGRE’s library, such as (i) TinyXML [18] (XML syntactic analyzer parser), (ii) OIS [19] (Interactions management), (iii) OgreAL [20] (Presentation of audio objects), (iv) DevIL [21] (Presentation of images inside VES), (v) Navi [22] (Presentation and interaction of a web browser inside the VE), and (vi) OGRE graphic engine which is the main module of the system being responsible for creating, managing and updating the tri-dimensional model;

(2) Elementary module, which describes the non-functional components of the architecture used as a support for the application (Meta-multimedia document and Virtual Environment);

(3) System startup, which is in charge to set up the presentation of multimedia content inside VES, and;

(4) System Update, which is in charge to control and update the multimedia presentation inside the VE according to possible user interactions.

The implemented prototype is further discussed on the next section.

![Figure 7. Architecture of OGRE-Multimedia](image-url)
4.3. An application of OGRE-Multimedia

This section illustrates the presentation of an OGRE-Multimedia application. This application applied a virtual world for the presentation of multimedia content in a showroom style. This environment can be applied for educational or art-exhibit purposes, for instance.

OGRE-Multimedia was easily integrated to the ogre application, where the methods of OGRE-Multimedia were invoked to enable the presentation of multimedia content previously defined on the meta-multimedia document, called “MMDocument.mmc”. In the case of this vr application, the virtual world describes an open-wide area, where in front of each wall there is a sensitive column which can be activated by the user’s click or by his proximity (depending on its previous configuration on the meta-multimedia document) in order to trigger the multimedia presentation on the wall (Figure 8a).

Figure 8(b) illustrates the interactive column in the ve. In particular, this column (which is represented by a trigger element on the meta-multimedia document) is able to launch a presentation by a user’s click.

The multimedia presentation is launched after the user interacts (by clicking) with the sensitive column (Figures 8c). When the media objects start to be presented, their level of transparency is gradually changed producing the effect of fading-in. As we can see in Figure 8(d), a web-browser is also presented as one of the multimedia textures. This browser is rendered by the api Navi which allows the user to navigate on the Web. Figures 8(d), 8(e) and 8(f) present an example of this navigation.

All the media objects presented inside the ve are synchronized and managed by OGRE-Multimedia which keeps pace of each presentation starting and interrupting all the media objects according to their previous configuration on the meta-multimedia document. OGRE-Multimedia enables the multimedia presentation inside the ve making of it a more realistic environment and, above all, keeping the user’s focus.

5. Conclusion and Future Work

This paper presents a broad view of two different efforts towards the customization of ves. These approaches share a common
technique which relies on the meta-language xml. The first approach proposes the customization of the characteristics of a virtual room and the 3D objects that may compose this environment using the Java3D API. The second approach proposes the customization of VEs with the presentation of synchronized multimedia content using the Ogre engine.

The first solution demonstrated the viability of customizing the main features of a VE. With XML it is possible to offer a simple interface for users to modify their application just by configuring the attributes of the environment and its 3D objects without changing a single line of the code.

The second solution presents Ogre-Multimedia which provides the customization of VEs with the presentation of synchronized multimedia content. This solution relied on the proposal of an XML-based representation to describe all the media objects to be presented and their synchronization relations. Ogre-Multimedia can be applied straightforward in different Ogre Virtual Reality applications since it is the result of an open-architecture where different APIs were applied in conjunction to provide the presentation of different kinds of media objects including the traditional images, audio, video, animations, etc., and also multimedia documents such as Flash, and web-browsers as well. Indeed, the combination of Multimedia and Virtual Reality can be successfully applied to the design of robust applications where users feel more comfortable and have their focus inside the VE, definitely improving their feeling of immersion.

Both approaches demonstrated that XML can be applied successfully as an intuitive interface for describing different features of the VE. Indeed, this interface can be generic enough for putting together different methodologies and techniques for authoring VEs. XML provides novice and advanced users with the capability of creating and customizing VEs dynamically without programming in a flexible and easy way.

6. Acknowledgments

We would like to thank Roberto Ivo C. de Freitas, Gonçalo Nuno P. Cardoso and Guilherme Fernando de Oliveira who have been actively involved in these projects.

8. References


