Semantically Enhanced, Multifaceted, Collaborative Access to Cultural Heritage

Project IST-034984

D4.2 Virtual Expedition Framework and Tools

Deliverable

Editor:	XLab
Contributors:	XLab, Technion
Dissemination Level:	Public
Version:	1.0
Date:	30-Jun-08

Executive Summary

Document Objective

This document describes the technical approach towards implementation of the Virtual Expedition Methodology, described in the Deliverable 4.1.

Targeted Audience

MOSAICA Consortium

Prerequisite

Deliverable D4.1 Virtual Expedition Methodology v2.2

Summary

This document describes the technical approach towards the implementation of the Virtual Expedition (VE) methodology, described in the Deliverable 4.1. VEs are envisioned as *one of the emerging technologies within the Web 2.0 paradigm of the Internet use.* It is thus of utmost importance to use modern and sound technological base for the development of the Virtual Expedition Framework. Furthermore, the suitable framework selection is not only guided by the technological needs, but also according to new and influential paradigms in the ICT community, like free and open source software (FOSS for short).

The document thus describes the reasoning behind technological choices, the actual implementation and the setup of the framework.

© MOSAICA

This document contains proprietary information of MOSAICA and shall not be reproduced or transferred to other documents, disclosed to others, or used for any purpose other than that for which it is furnished, without the prior written consent of MOSAICA. The trademark and service marks of MOSAICA, including the MOSAICA mark and logo, are the exclusive property of MOSAICA, and may not be used without permission. All other marks mentioned in this material are the property of their respective owners

Contents

1. Introduction 4
2. Brief Virtual Expeditions Requirements Overview 7
3. Technical challenges 9
3.1. Available frameworks 9
3.2. Brief GWT Technical Overview 12
4. Overview of the Virtual Expedition Framework 14
4.1. Communication Overview 14
4.2. VE Framework Components 15
4.2.1. The VEEditor Component 16
4.2.1.1. The VEToolbox 16
4.2.1.2. The WhiteBoard 17
4.2.2. The VEPlayer Component 17
4.2.3. The VEEditor and VEPlayer components combined 18
4.3. The VE Presentation Components 19
4.3.1. Events 22
4.4. Real Time Dialog (Instant Messaging) 23
5. Conclusions 27
6. Appendix A: VE XML description example 28
7. Appendix B: List of 3rd Party Tools and Libraries 29
8. Appendix C: MOSAICA Web API XSD 31
9. Appendix D: MOSAICA Application Properties file 33
10. Appendix E: MOSAICA DB creation schema 34

1. INTRODUCTION

MOSAICA is envisioned as a Web browser-based application. This design decision ensures that MOSAICA will operate in a system agnostic mode and that in such way we can make it available to the widest possible audience.

Following this design decision, the Virtual Expeditions (VE) must also run inside a Web browser. Moreover, VE has its own complex dynamics that was defined in D4.1 Virtual Expedition Framework, and that we shall present in practical terms in 2 Brief Virtual Expeditions Requirements Overview below. The consequence is that the implementation of the VE Framework must cater for creation of dynamic content within a Web browser. An application meeting this requirement can be easily compared to the so-called *mashups*[1]. – Mashup is a web application that combines data from multiple sources (usually located on distinct servers and belonging to different owners) into a single integrated display[2]. Mashup allows Web developers to capitalize on the existing Web applications by adding additional services to them, and by bundling them together to provide new functionalities. A typical mushup consists of three elements[3]:

1. **Service provider** – this is the basic application that provides API (usually through the Web Service interface), so that other applications can call its services, and incorporate them into a new application.
2. **Hosting application** – this is the server-side application utilizing the services provided by the service provider, and adding more functionalities to them. A hosting application can combine several service providers[4].
3. **Browser client application** – this is the executive side of the application running in a browser on the end-user's machine, and bundling together services provided by the hosting application.

MOSAICA Virtual Expedition actually follows closely this paradigm. In this case, the service provider is the *MOSAICA Semantic Layer*. Although both applications are co-developed within the same project, from the architectural point of view they are independent, and the interoperability is accomplished through API[5]. Moreover, the content used by the VEs originates from the remote locations on the WWW, although it is accessed through its URI, and not obtained from a third party via API.

---

1 http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

2 Mashup originally referred to the practice in pop music (notably hip-hop) of producing a new song by mixing two or more existing pieces together.

3 Adopted and adapted from http://www.ibm.com/developerworks/xml/library/x-mashups.html.

4 For example, Flickrvision (http://flickrvision.com/) combines Google Maps (http://maps.google.com/) and Flickr (http://www.flickr.com/) that lets you see the location on Google Maps of new pictures posted to Flickr as they are uploaded.

5 Sometimes it is expected these API to be public, in order to qualify for a true mashup.

The backend of the VE Framework serves as the mashup hosting application. This is where the content and data are bundled together into a single display. This is also where the specific additional functionalities are overlaid on top of the content. Actually, this backend utilizes the services provided by the *Semantic Layer*, in order to organize and integrate the content retrieved from the WWW into a coherent, meaningful whole.

Finally, the frontend utility applications included into the VE Framework (see 4.2 VE Framework Components) provide the browser-resident application that executes the mashup.

It is worth noticing that MOSAICA was formulated for the first time as a project proposal, less than a year after the term *Web 2.0* was coined[6] by Tim O'Reilly[7], in order to describe a wave of novel web applications overtaking the WWW. The term *mashup*, borrowed from the popular culture, started appearing around that time to describe the new technological approaches and the different Web architectures that started appearing on the WWW. – For example, one of the pioneering mashup websites, http://chicago.everyblock.com/crime/ appeared in 2005[8]. The influential IBM's website for mashup developers (http://www.ibm.com/developerworks/xml/library/x-mashups.html) was launched only in August 2006, already few months after the MOSAICA's kickoff. At the time when the Virtual Expeditions were first envisioned it was clear what would be the desired outcome. However, the terminology and the paradigm were still missing, and it was not clear how to implement this vision technologically. As the Web 2.0 matured, the technological solutions became more obvious, and a generation of new technological tools appeared. MOSAICA both benefited from these developments and positioned itself into the frontrunner's position in the modern-day WWW. Nevertheless, the novelty of the solutions added a significant level of complexity (and uncertainty regarding the outcome) to the implementation of the VE Framework.

To demonstrate how novel are these technologies, it is sufficient to mention that Intel recently released Mash Maker[9] (http://mashmaker.intel.com) that is designed to provide a non-technological user with the ability to build mashup websites (under the motto of "mashup for masses") – an event that is comparable to the first appearance of the WYSWYG HTML editors.

The VE Framework browser-based applications (the third component in MOSIACA mashup) turned out to be in particular complex for implementation. While mashups simply display output originating in other applications, MOSAICA VEs must in addition to that also support events. The reason is the highest level of interactivity required from the VE Framework.

6 http://www.paulgraham.com/web20.html
7 http://web.archive.org/web/20040606020305/www.web2con.com/
8 http://en.wikipedia.org/wiki/Adrian_Holovaty
9 http://www.pcworld.com/article/id,144926/article.html

This was even further complicated by the requirement to enable the MOSAICA Web application in general to be adaptable by various interests groups to their specific needs. Accordingly, the obvious choice was to insist on the use of *Free and Open Source Software* (FOSS) development frameworks and tools, thus enabling further work on the MOSAICA project output by the communities.

2. BRIEF VIRTUAL EXPEDITIONS REQUIREMENTS OVERVIEW

This section describes and summarizes the requirements for VEs, as given in D4.1, Section4.3.

The VE typology overview, presented in D4.1, Section 4.2, provides an insight into the complexity of the technical solution that needs to be developed. The following is a brief summary of the requirements (D4.1, Section 4.3):

- **Flexibility** – must support a very broad range of potential topics (D4.1 presented typology comprising seven different topic-based types).
- **Adaptability** – must support a variety of plots and narratives.
- **Support of Creativity** – must be capable of supporting different genres and approaches.
- **Appeal** – must allow authors to develop a high appeal by providing unique, distinguished overall conceptual design.
- **Presentational Variety** – must support a variety of presentation and narrative techniques, ranging from dialog to quest to exploration. Based on computer gaming, cinematography and graphic novels, D4.1 identifies 5 different narrative types: dialog (telling the story through conversation between characters – including the user), detection (telling the story by letting the user discover its elements), use (telling the story by making the user actively reconstruct it – for example, by using objects), problem solving (telling the story by solving problems that constitute the plot), decision making (telling the story by transforming the user into one of the characters, and allowing the user to take decision in the story).
- **Novel User Experience** – must provide a novel user experience, different from the current WWW standards, and based on the curiosity, challenge and feedback.
- **Collaboration and knowledge sharing** – users must be allowed to contribute privately owned content, and Virtual Expeditions must offer means for informal discourse among users.

Given the above requirements it is evident that the development tools or development toolkit required for implementation of such constraints, needs to be, from the technical standpoint, rather low-level. In other words, the development toolkit must provide the developer with sufficient freedom in order to develop the adequate framework.
D4.1 also provides guidelines for the development of the necessary components for VEs, and detects the following controls that need to be implemented:

- **Resource Control** – displays either text, or an external multimedia file, or assertions from an ontology, or combination of these – in an **integrated** display

window.

- **Mosaic Control** – an extension of the Resource Control, providing implementation of the mosaic action prompt: marking of specific object areas on the displayed resources, and attaching separate menus to these object areas.
- **External Resource Control** – used to invoke an external resource in a **separate** display window.
- **Grab and Drop Control** – used to collect and relocate objects within the environment.
- **Trigger Control** – activates another object in the environment.
- **Dialog Control** – provides a gradual exposition of the information in a form of a dialog with a virtual character.
- **Real Dialog Control** – used to exchange information with other users in real-time while executing a Virtual Expedition (it is an instant messaging control).
- **Quiz Control** – used to prompt user with a task, question or problem.

Each of the listed controls must provide an interface in the form of a property dialog box used for the configuration of the action settings (similar to the implementation of controls and widgets in the Integrated Development Environments). The accurate description of the above controls is provided in D4.1, Section 5.2.2 Toolbox. Here, we shall just emphasize that the requirements for these controls, in terms of interactions between them, raise the need to implement a flexible, event driven and completely customizable online development tool for the end-user.

3. TECHNICAL CHALLENGES

In order to start developing the VE Framework the MOSAICA consortium had to decide on a number of issues, most notably on:

1. The issue of having a two- or three-dimensional VEs,
2. The accessibility of VEs on major platforms (from the development, creation and presentation point of view).

We present the reasoning behind our decisions and overview of the technology below.

3.1. Available frameworks

The main decision which framework should be used was to decide upon two- or three-dimensional VEs. As this decision was based upon the availability of the existing frameworks and on the expected user-base of the MOSAICA project, we had to evaluate a number of different frameworks which could cater the needs of the pedagogical approach towards Virtual Expeditions.

Let us first explore the possibility of using a 3D virtual environment for VEs. The common agreement between the consortium partners is that the 3D virtual environments are more "attractive" for the end-user. Following this criterion of attractiveness we started exploring the 3D environment offered by Caspian Learning[10] - Thinking Worlds. They offer a complete pedagogical framework which is used in various companies for various employee training. While embracing such a complete solution for VEs was at first very appealing, review of its properties soon revealed its shortcomings:

- Creation
  - The environment in which the VEs were to be created is of limited flexibility. While such approach works very well in the model where the company itself releases the e-learning programs, the VEs in MOSAICA are envisaged to be very flexible.
  - The creation of various environments is costly, if developed by the Caspian Learning.
  - Graphical objects are created in 3D Studio Max[11], which is neither free nor simple to use. The creation of such graphical objects would require skilled graphic designer, thus limiting the content creation only to them.
  - The design environment consists of an offline program (in addition to the 3D Studio Max) which also violates the requirements for MOSAICA VEs.
  - The integration with the MOSAICA services would be challenging.

[10] http://www.caspianlearning.co.uk/

[11] http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=5659302

- Presentation
  - The already made presentations are available via ActiveX[12] control. This seriously limits the access to the VEs, as the aforementioned controls can be run only in Internet Explorer, available only on the Microsoft Windows platform.
  - Following this exploration we have found out that while 3D environment may be more attractive, it presents the end-user with difficulty of creating the actual 3D objects and hence reduces the user-base which could contribute to creation of Virtual Expeditions. Furthermore, another obstacle is the Intellectual Property Rights (IPR from now on), which limits the modification of the existing content to next to none. In practice this means that the existing content (or content available through signed contracts) cannot be exploited in 3D environment as it would have to be modified in order to be grafted into 3D scenery. Use of such existing and unmodified content in MOSAICA would result in rather unconvincing 3D virtual reality[13].

To sum up, due to predicted reduction of the expected user base for creation of Virtual Expeditions and due to restrictions imposed upon use of existing resources on the web (IPR issues), the MOSAICA consortium agreed to use simpler, 2D environment.

Following this change, we started evaluating and testing the available technologies which could potentially offer technological development tools for convincing presentation of the Virtual Expeditions under the above described constraints.

The overviewed and tested technologies were:

- Complete solutions:
  - Blaxxun (www.blaxxun.com)
  - SOS-21 (http://www.sos-21.com)
- Base technologies:
  - Adobe Flash (http://www.adobe.com/products/flash/) (along with FOSS Flash compilers)
  - Adobe FLEX (http://www.adobe.com/products/flex/?promoid=BPDEQ)
  - Google Web Toolkit (http://code.google.com/webtoolkit/)

Let us first describe the two complete solutions. They both offer either 2.5D or 3D virtual environment thus almost the same problems as with the Thinking Worlds are relevant. Still, the SOS-21 technology is rather simpler (2.5D) and offers online creation of the presentation. Also, the creation of the content is simpler (due to 2.5D) and requires only a good graphic editor (e.g., GIMP[14], licensed under GPL). The evaluation of the integration with the MOSAICA services would present a challenge, but could be done.

[12] http://activex.microsoft.com/

[13] We support this thesis with simple example: using images, available from different sources, means that they are presented in rather different color palettes, light, etc. Use of such material is acceptable and common in "ordinary" web page, but not in the 3D virtual environment.

[14] http://www.gimp.org/

The problem with the SOS-21 technology (we're limiting ourselves just to description of this one as it had better potential than Blaxxun, which offers 3D) is that it also has a proprietary back-end in which it stores the actual presentations. This would lock the MOSAICA project to one vendor and limit the use of project output.

The base technologies which are now used for site development are numerous, ranging from simple HTML to Silverlight[15]. In order to satisfy the requirement of using the development platform which is available for all major operating systems and which enables the user or developer to run the MOSAICA application on all major operating systems, we limited ourselves to the products of the Adobe and Google company. These two companies offer at least free software development kits (SDK from now on) or there exists a FOSS compiler for their technology. Let us first focus on Adobe Flash and Flex technologies. For Flash, there exist free compilers, which can be utilized from the simple web page and would produce the Flash animation. Our tests with such setup revealed that it is too unstable and ultimately inflexible for serious use in MOSAICA. For FLEX we can obtain a complete and free SDK, but the tests of developing software in such a way (without purchasing the specialized tools) revealed that it is very clumsy. Still, we could use it, but the end-product would be inflexible.

In the light of these discoveries, we turned ourselves upon using GWT, which is free and offers integration with one of the best FOSS Integrated Development Toolkits – Eclipse[16]. The tests with GWT revealed that it is the most low-level of all of the evaluated tools thus it offers the flexibility we are seeking for MOSAICA development. On the other hand, it offers an elegant programming model (development is in Java, which is then compiled into JavaScript) which enables easier support and maintenance. Furthermore, we discovered the extension of the GWT, called GWT-Ext[17], which combines one of the most powerful JavaScript libraries Ext with GWT and provides the Java wrapper.

To sum up, the chosen development tools and hence technology is Google Web Toolkit. We support this decision with:

- The development tools and the SDK are FOSS, hence anyone can use them to add new functionality to the MOSAICA project even after the project is finished.
- The development tools are integrated into Eclipse, the most prominent FOSS IDE.
- GWT offers dynamics, needed to create appealing online editing and playing tools.
- GWT technology offers best integration possibilities among the evaluated tools.

15 http://silverlight.net/
16 http://www.eclipse.org
17 http://code.google.com/p/gwt-ext/

3.2. Brief GWT Technical Overview

In this section we turn to the GWT in the sense of describing where the developed application runs, as this description is of utmost importance for understanding the following descriptions of the VE Framework.

GWT is using the well known servlet[18] technology, hence it follows that the developed application (in our case, the MOSAICA application and the VE Framework) are running in the client browser. The requested data is obtained from the application server via Remote Procedure Calls (RPC for short).

So, to deploy GWT application one needs an application server, capable of serving Java servlets – in our case, the selected application server is the Apache Tomcat Server[19] (Tomcat for short). At this point we must note that it is possible to use other Java servlet servers and we also may have to change the Tomcat server due to its internal threading model.

When a user requests the MOSAICA application (and, implicitly VE Framework), the application is loaded into his browser (the application is running on the client-side in the browser). When additional data is needed, the application contacts the appropriate servlet which asynchronously returns with the requested data. The described procedure is thus client-server architecture with asynchronous remote procedure calls (i.e., the calls to the server can also be described as non-blocking).

At this point we observe the first pitfall of the GWT in MOSAICA VEs – such asynchronous calls to the server-side provide us only with one-way communication, while VEs request two-way communication due to at least one control – Real Time Dialog Control (described in later sections).

In order to provide the solution for this requirement, we exploit the Comet paradigm, which is just a common name for opening a constant stream of data from server to the client browser. To this end, we used the implementation of the aforementioned paradigm called Rocket[20], which is already integrated with the GWT. Furthermore, to use the same connection for different kinds of data (not just the Real Time Dialog Control) we implemented multiplexer/demultiplexer mechanisms on the client and server side.

To sum up this brief GWT overview, we just note that the calls to the server (server denotes our Java servlet server – Tomcat) are asynchronous and that we are able also to communicate from server to the specific browser and thus, to the specific client.

18 http://java.sun.com/products/servlet/
19 http://tomcat.apache.org/
20 http://code.google.com/p/rocket-gwt/

4. OVERVIEW OF THE VIRTUAL EXPEDITION FRAMEWORK

In this section we describe the overview of the VE Framework, from the communication with various MOSAICA services to the actual components and the mechanism between them.

Before we start with the actual descriptions, we must raise the issue of VE control in terms of traceability of the user. Namely, the MOSAICA project envisages moderating system which helps with detection of the objectionable content. This issue is first and foremost solved with the requirement of logging into the system, before we can actually create VEs, comment them, etc. This means that the users who are creating or commenting VEs can always be banned from MOSAICA by means of their username and/or their respective IP.

4.1. Communication Overview

VE Framework needs to exploit the MOSAICA application communication model. The communication between the client (browser) and server (Java servlet server) has already been briefly described above, therefore we present the communication diagram for the whole VE Framework and MOSAICA application (as schematically depicted in Figure 1).

The diagram shows two users connected to the MOSAICA application, using VE Framework. Technically, the red arrows denote ordinary GWT asynchronous calls to the server, while the blue arrows denote the open connection towards the browser, implemented with Rocket software stack. The black arrows denote Web Services calls to other MOSAICA services (GIS/TIS, p2pCMS, etc.).

It is important to note that we have two sets of arrows for two VE Framework users – they are distinguished by their HTTPSessions (and thus by corresponding Session Ids) thus both users have access to their "state". Technically, the state of application is stored in the browser session, but can be also stored at the server-side and also used in the MOSAICA services in order for them to provide different sessions for individual users. Let us explain this important fact on a simple example, using the Semantic Browsing component. When a user is searching through the MOSAICA heritage items, using Semantic Browsing component, this component runs on a different server than the MOSAICA application and also MOSAICA VE Framework. The communication between the servers (inter-server communication) is accomplished using Web Services (HTTP Rest). It logically follows that because the Semantic Browsing component has its

own GUI, it needs to now what to present to an individual user, thus the user ID (or, equivalently, session ID) must be used to present the proper data to the user. So, having separate sessions for individual users and using this data for other stateful services is technically unavoidable. The XML Schema Definition for the calls between web services is shown in Appendix (see Table 5 MOSAICA application web API XSD).

Having covered the communication between the VE Framework, the Java Servlet server and the MOSAICA services, we continue with the overview of the VE Framework itself.

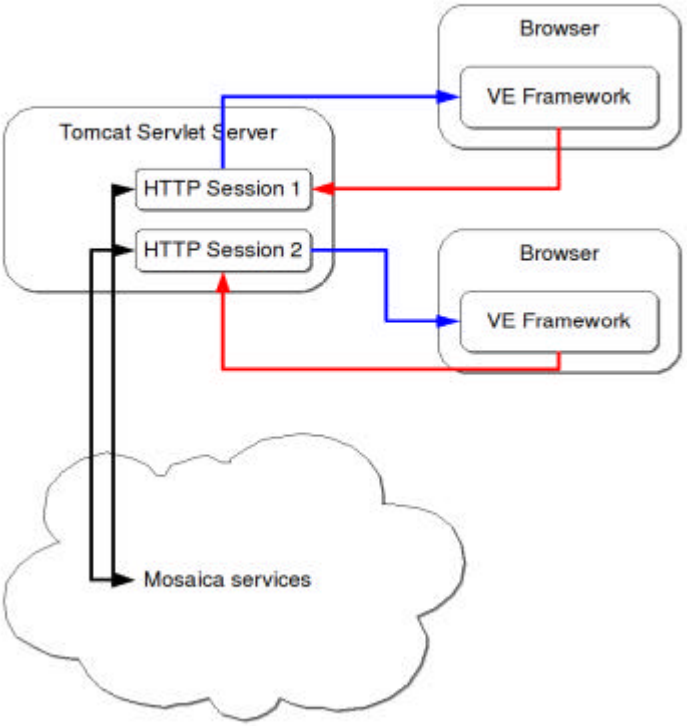

Figure 1 Communication diagram from the VE standpoint

4.2. VE Framework Components

The VE Framework components are divided intro three types:

1. The VEEditor
2. The VEPlayer
3. The VE Presentation Components

These components constitute the three main pillars of the VE framework. The components that surround them (the so-called helper components) are:

4. Server located database
5. VE commenting component
6. VE rating component
7. XML utilities for import and export of the VEs

These components will be described in the following sections.

4.2.1. The VEEditor Component

The VEEditor consists of the WhiteBoard and the VEToolbox components which represents a large part of the infrastructure needed for editing and testing of the VEs. The below images graphically present these components, which will be better explained in the following text.

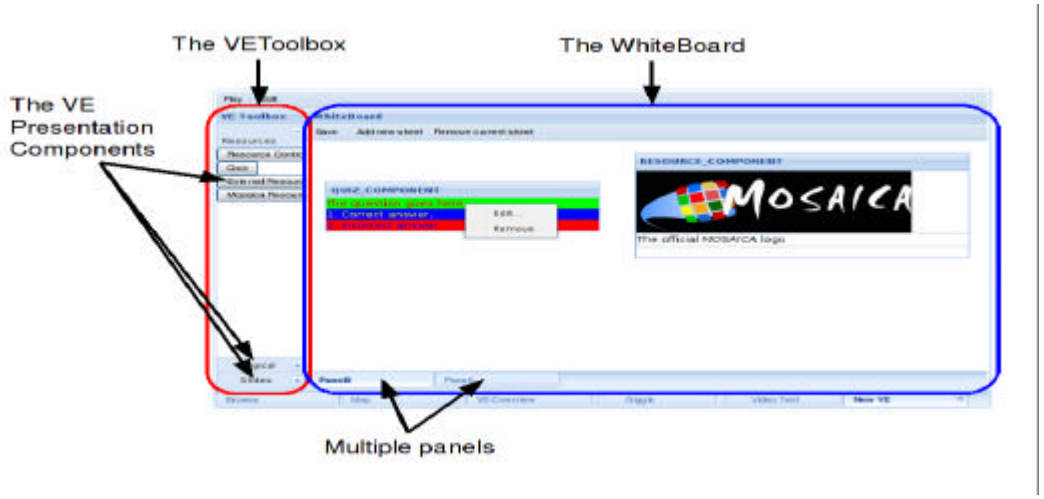

Figure 2 Main VEEditor window

4.2.1.1. The VEToolbox

The VEToolbox consists of the components described above, in the VE Framework Requirements. The only exception is the Real Dialog Component, which is located on the main MOSAICA application window in order to facilitate communication between MOSAICA users regardless of them exploring or creating Ves. The IM component is always available, but can be collapsed (i.e., hidden) if there is a need for larger desktop space.

As you can also observe, there are only four component buttons shown under the common tab Resources – others are hidden under tabs Logical and Slides. When the user wants to add one of the components, he just clicks on the respective button and the component appears in the WhiteBoard.

4.2.1.2. The WhiteBoard

The WhiteBoard component one of the most complex components in the VE Framework infrastructure. Figure 2 shows the component rounded with a blue square. The component handles the creation and destruction of the VE Presentation Components and also implements load and save methods, where the Save button is visible, while loading is implicit via creation of new VE or via loading of an existing VE, which are listed in the main MOSAICA application tabs (the tab with a name VE Overview).

Load and save methods actually import and export the components that are put on the WhiteBoard into or from XML description – one rather self-explanatory example of the VE described in XML is shown in Appendix – see Table 4 Prototype of the XML VE description. It must also be noted, that the user is asked for his description of the VE (in free text) before storing of the VE. We can then search through these descriptions via FullText Search in the VO Overview tab of the main MOSAICA Application.

The WhiteBoard also enables the user to add multiple sheets (one can think of them as slides) and therefore allowing for creation of complex and large VEs.
Finally, the WhiteBoard also enables positioning and rearranging of the components which are created on it.

4.2.2. The VEPlayer Component

The VEPlayer consists of only the VEPlayerBoard, which is essentially scaled-down version of the WhiteBoard component. While the WhiteBoard component allows rearranging the components, the VEPlayer doesn't allow that. It also does not implement XML export methods, as the VE cannot be changed when played.
With activation of a VE with the VEPlayer, three additional components become available (as schematically depicted in Figure 3):

- **Grading VEs:** This feature is widely known as the Rate-it feature. The user has the ability to grade the VE he has just played with mark ranging from 1 to 5 (with moving the mouse cursor over the stars from left to right). This grade is stored and used for calculation of the average grade – shown in the VE Overview tab (part of the MOSAICA application).
- **Commenting on VEs:** The user can always comment on the VE he has just viewed or just see the comments of the previous users.
- **Reporting content:** If the user finds the VE objectionable, he can always report the content to the MOSAICA administrator who can, using the username of the irresponsible person, simply ban the person from the MOSAICA VE Framework and change/delete the offending content.

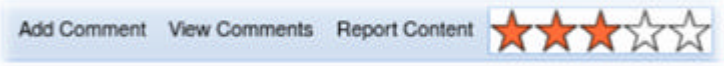

Figure 3 Toolbar for grading and commenting on VEs

4.2.3. The VEEditor and VEPlayer components combined

Before we present the interfaces and code for the VE Presentation Components, we must describe the collaboration between the VEEditor and VEPlayer. Namely, the intention of having an online VE Framework is to provide the round-trip from creation to presentation with which the user can check the created VE before he saves it and thus enables access to it to the other users. This is achieved through local (in the browser) XML import and export procedure: the VE is exported from the VEEditor into XML (locally) and then immediately imported into the VEPlayer. The user can thus see whether he actually did set up all the properties and connections between the VE Presentation Components properly – in essence, the user is presented with a lighter version of the WYSIWYG Rapid Application Development. The implementation of the latter was envisaged in order to help inexperienced users in their trial and error process of learning the new environment.

Another feature is extension of the existing VE. Namely, if some user already created an interesting VE, another user can add new and possibly more in-depth content in order to improve the VE. Still, the VE Framework will hold and store both of the VEs in order to enable completeness of the created and contributed work (the users will always give better grades to better VEs thus rewarding additional work, if it is good).

Finally, we need to explain where the usernames and user IDs, XML descriptions, VE grades, comments, etc., are stored. To this end we use a central database, with creation schema shown in Appendix – Table 7 MySQL script for the MOSAICA server database. The decision to use such a central database despite having a p2p-CMS in the MOSAICA component repository was taken in order to improve parsing and loading times for VEs.

The database MOSAICAVirtualExpeditions consists of the following tables[21]:

- **VEMedia:** This table stores the data about the media files, which were uploaded to the MOSAICA system – these files will ultimately be stored in the p2p-CMS, but in order develop the VE Framework, these files are for the time being stored locally. Still, this table is not just for the development purposes – within the VEs, we will use it to cache the interesting media in order to keep the utilization of the p2p-CMS reasonably low. Further use of this table can be achieved by measuring the actual number of seeds in the p2p-CMS Bittorrent layer – the data with low number of seeders can be transferred to the main MOSAICA server to improve load times of the VEs.
- **VEDescription:** The table stores data that is relevant to the XML descriptions of the VEs. The field VEXML holds the XML description of the whole VE, while other fields enable traceability and other information regarding the VE.
- **VEGrades:** The name of this table is self-explanatory, as it holds the grades of VEs which are then averaged when the VE Overview tab requests the VE average grade.

[21] The properties that need to be set for the database are shown in Table 6 MOSAICA application properties file.

- **VEComments:** The user comments regarding VEs are stored in this table.

It is also important to note that the VEMedia table holds the transcoded data, as described in the D1.2. Namely, the development of a mechanism that enables simple presentation of multimedia content on the web, combined with the intellectual property rights management (IPR)[22], resulted in design and development of special module that is able to transcode the multimedia content. Special consideration was put to selection of the final presentation format, in order to prevent the installation issues with various codecs at the end-user machine[23].

Finally, the usernames are stored in the OpenFire database (see Section 4.4) which is responsible for the Real Dialog Component.

4.3. The VE Presentation Components

This section describes the technical insight into implementation of the VE Presentation Components (in this section we will call them simply components). All the components implement the following interface:

```
/**
 * Public interface for the VEEditor Components -> this interface must
 * be implemented by all the VEEditor components.
 * @author danielx
 */
public interface IVEComponent {

	public int[] getLocation();
	public void setLocation(int x, int y);
	public Size getSize();
	public void setSize(Size size);
	public HashMap getProperties();
	public void setProperties(HashMap properties);
	public String getComponentType();
	public ComponentEditor getEditor();
	public WhiteBoard getWhiteBoard();
	public void setWhiteBoard(WhiteBoard whiteBoard);
	public Panel getDisplayComponent();
	public String getId();
	public void handleEvent(IEvent event);
	public void fireEvent(String targetID, IEvent event);
	public void setEventDispatcher(IEventDispatcher ed);
	public Iterator getSupportedEvents();
	public IEventHandler getEvent(MOSAICAEvent event);
}
```

Table 1 The VEComponent interface.

While most of the methods are self-explanatory, we expose the following important methods:

- **get/set properties**: As already described, the WhiteBoard handles the overall VE XML import and export. Still, the WhiteBoard is component agnostic – during the import and

[22] They are explained in greater detail in the following paragraphs dealing with WP4.

[23] The properties that need to be set for this module are shown in Table 6 MOSAICA application properties file.

export phase, it asks the component to set and export its description in the XML format. This means that each component needs to implement its persistence methods in the MOSAICA VE Framework, resulting in straightforward extendibility of the existing set of components with new components.

- **get editor**: The user needs to handle components through the form-based dialog. As each component has a different set of properties, it is required to implement its own editor. Still, the ComponentEditor class, which needs to be extended with the particular component properties offers helper utilities (e.g., a method which gets all the components on the current panel), which simplify the development of new components.
- **handle/fire event**: These two methods cater for handling of events, received from other components or for firing events to other components.
- **get supported events** : Each of the components lists supported events or, in other words, the events it can handle. Examples of such events are: EventEnable, EventDisable, EventMove, etc., which enable, disable or move the component it was fired upon, respectively.

Having described the important interface methods, we proceed with elaboration of the example, already shown in Figure 2. The WhiteBoard holds two objects: the quiz and the resource control component. The latter is already configured to show the MOSAICA logo and text, while the quiz component is in its default state. With the right mouse click on the component, the menu with two options (Edit, Remove) opens. Selecting Edit invokes the getEditor method of the quiz component, shown in Figure 4. By setting the question and appropriate (multiple) answers, we can finish basic editing of the control (shown in Figure 5).

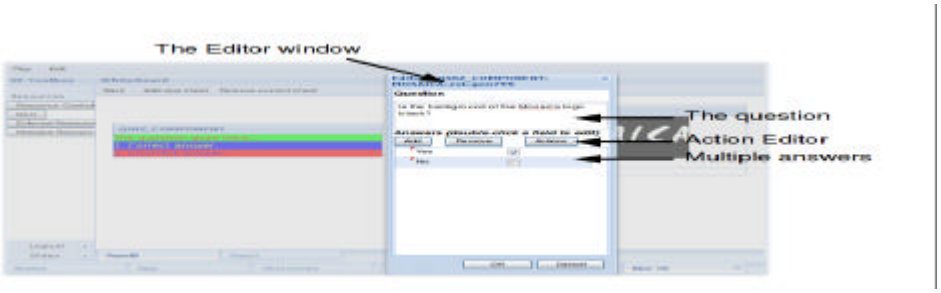

Figure 4 Editor window of the QUIZ component

At this point it is important to note the Actions button, which invokes the ActionEditor – a component, which enables editing of events that are fired to selected other components when a certain condition is satisfied. In the quiz component case, the events can be fired with clicks on various answers. In case of the current simple VE, we set up the actions that enable or disable the resource control (with the MOSAICA logo) based on validity of the clicked answer.

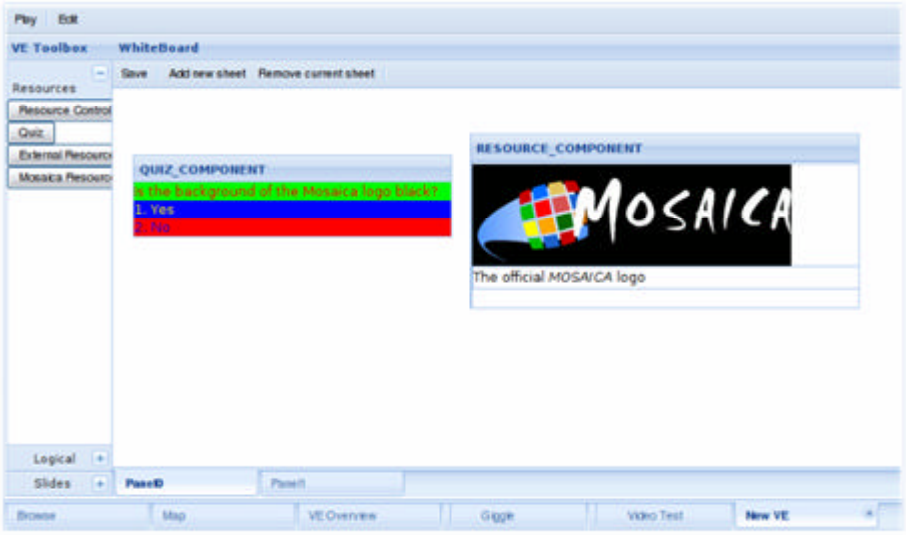

Figure 5 The final result of the QUIZ component editing

4.3.1. Events

As we have already explained at the beginning of this document, the VE Editor must enable the user to connect different components via events and hence define actions on different components, based on "triggers".

In the editing phase of the VE, we exploit the fact that the components themselves define which events they are able to handle – this is a fixed set of events for a particular component. The ActionEditor component, when invoked on a particular component, gathers all the components over all panels and extracts their supported events. The user can then simply decide which actions on the component he is currently editing trigger supported events on other components. Furthermore, one action can trigger an arbitrary number of events on one or multiple components.

In the presentation phase of the VE, the actions on components fire events to other components and below are the excerpts from the interfaces through which we achieve such behavior.

The interfaces presented in Table 2 are the main interfaces for implementation of the event manager. While the implementations of the IEvent interface hold all the data about the events and can be easily extended with new types of events, the IEventDispatcher implementation is final and implements the storage and invocation mechanism for the target components and their respective handleEvent methods.

```
public interface IEventDispatcher {
	public void registerComponent(IVEComponent component);
	public void unregisterComponent(IVEComponent component);
	public void dispatchToComponent(String componentID, IEvent e);
}


public interface IEvent {
	public int getEventID();
	public String getEventName();
	public Object getEventData();
	public String getEventTarget();
}
```

Table 2 Main event interfaces

We finally need to provide a technical reasoning behind the implementation of such an event management mechanism. We tested the native JavaScript mechanisms of firing and consuming events, but they proved to be unreliable – we attribute this behaviour to specific needs put upon the Java to JavaScript compiler used withing GWT.

Furthermore, by use of custom and simple event management mechanism, we achieved decoupling of method invocation between components in VEs, which is beneficial for further improvements and support of the VE Framework.

4.4. Real Time Dialog (Instant Messaging)

This section presents the IM component, embedded into the MOSAICA application which was first envisaged as an component for the VE Framework. As it is responsible for the large part of technical research in the VE Framework development, it is our opinion that it deserves a separate section.

In order to implement the Real Time Dialog control (i.e. Instant Messaging within a browser, which is called IM from now) we had to use a protocol for communication between clients. The decision between implementing a new protocol for instant messaging or using a pre-existing protocol was rather trivial to solve. As there are a plethora of existing IM protocols (e.g., ICQ, MSN, XMPP, ...) we decided to use the XMPP[24] protocol (also known as Jabber).

We support our decision for selection of the XMPP protocol with the following reasons:

- It is open source.
- It is formalized by the Internet Engineering Task Force (IETF [25]) in the RFC.
- There exists a plethora of existing (offline) applications allowing for connections with this protocol.
- There exists a GPL version of the libraries that are needed for its successful use in proprietary implementation of the IM client.

[24] http://www.xmpp.org
[25] http://www.ietf.org

The GPL libraries (i.e., the software stack) for implementation of the IM client selected is *Smack*[26], which is known to be robust and rather easy to use. Below (see Table 3) are examples of use from Java (code excerpts from MOSAICA project). We can see that the basic use of the Smack software stack for operating the IM control is straightforward. Still, in MOSAICA we use additional methods in order to improve the detection of user presence, security, etc.

```
public class MOSAICAXMPP {
	private static final Log log = LogFactory.getLog(MOSAICAXMPP.class);

	XMPPConnection cnx;
	MultiUserChat muc;
	String user;

	PushQueue queue;

	public MOSAICAXMPP(PushQueue queue) {
		this.queue = queue;
		log.debug("MOSAICAXMPP interface created with push queue.");
	}

	//Open connection to the OpenFire server
	public void open() throws XMPPException {
		MOSAICAConfig cfg = MOSAICAConfig.getMOSAICAConfig();
		ConnectionConfiguration config = new ConnectionConfiguration
			(cfg.getXMPPServer(), cfg.getXMPPPort());

		cnx = new XMPPConnection(config);
		cnx.connect();
		cnx.addPacketListener(new MOSAICAChatListener(),
			new MOSAICAChatFilter());
		cnx.addPacketListener(new MOSAICAPresenceListener(),
			new MOSAICAPresenceFilter());
		muc = new MultiUserChat(cnx, getMOSAICARoomJid());
		log.debug("MOSAICAXMPP connection open.");
	}


	//Login to OpenFire XMPP server (group mosaica)
	public void login(String user, String pass) throws XMPPException {
		cnx.login(user, pass, "mosaica");
		muc.join(user);
		this.user = user;
	}

	//Send message msg to user with user ID jid
	public synchronized void sendMessage(String jid, String msg) throws
ChatException {
		Message m = new Message(jid, Message.Type.chat);
		m.setBody(msg);
		cnx.sendPacket(m);
	}

	public void close() {
		cnx.disconnect();
	}
}
```

Table 3 The most significant methods for operating the IM control

Furthermore, in order to provide complete independence from the free XMPP providers, the MOSAICA project uses its private XMPP server – OpenFire[27], which is also known

[26] http://www.igniterealtime.org/projects/smack/index.jsp

[27] http://www.igniterealtime.org/projects/openfire/index.jsp

to be very robust and easy to install and administer. The description of installation of the MOSAICA XMPP server is beyond the scope of this document and also described online[28]. The XMPP server properties (IP and port) are described in the `mosaica.properties` file, shown in Table 6 MOSAICA application properties file.

Finally, we are presenting an image of the actual control. It consists of four different panels, where (from top to bottom) the first one consists of the list of the all users, currently logged into MOSAICA, the second one shows the persons we (the user with this particular chat window) is chatting with, the third one shows the chat log and finally, the fourth one enables the user to type his message to another user. The user interface of this control is thus equivalent to Pidgin[29] user interface, which is widely used, hence the users are familiar with it.

Figure 6 The annotated IM control

28 http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/install-guide.html

29 http://www.pidgin.im/

5. CONCLUSIONS

In this document we described the most important aspects of the VE Framework development and its internal structure. While the main objective was to satisfy the requirements described in D4.1, we also tried to satisfy additional requirements which are:

- Clean architecture of the VE Framework, enabling future development and easier support.
- Decoupling of the main components enabling simple extendibility of the VE Framework.
- Reliability and detachment from the bugs of the underlying development framework.
- Use of FOSS libraries and development tools.
- Enabling use of the project output by the widest possible audience.

6. APPENDIX A: VE XML DESCRIPTION EXAMPLE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ve>
    <panel>
        <object>
            <Description>
                <Text>&lt;p&gt;Some text here&lt;/p&gt;</Text>
                <OntologyMethod>none</OntologyMethod>
                <URI>http://www.xlab.si/images/xlab-logo.png</URI>
            </Description>
            <ComponentID>MOSAICA-ext-gen736</ComponentID>
            <Location><Y>78</Y><X>0</X></Location>
            <ComponentType>RESOURCE_COMPONENT</ComponentType>
            <Size><Height>400</Height><Width>600</Width></Size>
        </object>
        <object>
            <Description>
                <Question>The question goes here...</Question>
                <Answers>
                    <Correct>true</Correct>
                    <Action>
                        <EventName>EventEnable</EventName>
                        <EventTarget>MOSAICA-ext-gen736</EventTarget>
                        <EventID>2</EventID>
                    </Action>
                    <Answer>Correct answer.</Answer>
                </Answers>
                <Answers>
                    <Correct>false</Correct>
                    <Action>
                        <EventName>EventDisable</EventName>
                        <EventTarget>MOSAICA-ext-gen736</EventTarget>
                        <EventID>3</EventID>
                    </Action>
                    <Answer>Incorrect answer.</Answer>
                </Answers>
            </Description>
            <ComponentID>MOSAICA-ext-gen895</ComponentID>
            <Location><Y>0</Y><X>0</X></Location>
            <ComponentType>QUIZ_COMPONENT</ComponentType>
            <Size><Height>78</Height><Width>300</Width></Size>
        </object>
        <panelID>0</panelID>
    </panel>
</ve>
```

Table 4 Prototype of the XML VE description

7. APPENDIX B: LIST OF 3RD PARTY TOOLS AND LIBRARIES

1. **Rocket GWT library/framework** consists of a number of components that can assist developers get more out of GWT. A variety of modules exist each performing a specialized task, which are at various levels of maturity. Furthermore, most of the modules duplicate functionality provided in GWT-Ext at a lower degree of maturity, hence we use only the **rocket.remoting.Remoting** package, which provides support for the Comet paradigm (server push), JSON RPC, Java RPC ((using rocket's serialization). Other useful (but currently not completely stable) modules/widgets:
    - **Basic widgets** - Does everything the equivalent GWT widgets plus also accept Elements from the DOM.
    - **Template based widget and panel authoring**. To hijack existing elements, sinking events, styling just override the appropriate method.
    - **Calendar** - A easy to use template based Calendar widget. Easy to right your own DatePicker etc.
    - **HtmlTemplateFactory** - hijack elements from the DOM, embed files, execute templates on the client.
    - **ImageFactory** - smart use of data urls for images whenever possible.
    - **Menu** - includes a context menu, control which way lists open up, down, left and right.
    - **ResizablePanel** a panel that has draggable edges that stretch the widget its holding.
    - **Slider** - slides which include support for a background widget.
    - **SortableTable** - a multi column sortable table that maps value objects to table rows.
    - **TabPanel** richer tab panels widget.
    - **Viewport** - create a Google map tile based draggable viewport.
2. **Openfire** (formerly Wildfire) is a real time collaboration (RTC) server dual-licensed under the Open Source GPL and commercially. It uses the only widely adopted open protocol for instant messaging, XMPP (also called Jabber). Openfire is incredibly easy to setup and administer, but offers rock-solid security and performance.
3. **Smack** is an Open Source XMPP (Jabber) client library for instant messaging and presence. A pure Java library, it can be embedded into your applications to create anything from a full XMPP client to simple XMPP integrations such as sending notification messages and presence-enabling devices.
4. **MySQL** is the world's most popular open source database that became the database of choice for a new generation of applications built on the LAMP stack

(Linux, Apache, MySQL, PHP / Perl / Python.) MySQL runs on more than 20 platforms including Linux, Windows, OS/X, HP-UX, AIX, Netware, giving you the kind of flexibility that puts you in control.

5. **Google Web Toolkit (GWT)** is an open source Java software development framework that allows web developers to create Ajax applications in Java. It is licensed under the Apache License version 2.0.
6. **GWT-Ext** is a powerful widget library that provides rich widgets like Grid with sort, paging and filtering, Tree's with Drag & Drop support, highly customizable ComboBoxes, Tab Panels, Menus & Toolbars, Dialogs, Forms and a lot more right out of the box with a powerful and easy to use API. It uses GWT and Ext 2.0.2 LGPL.

8. APPENDIX C: MOSAICA WEB API XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="http://research.xlab.si/namespaces/mosaica/2008/06/webapi"
    xmlns="http://research.xlab.si/namespaces/mosaica/2008/06/webapi"
    xmlns:tns="http://research.xlab.si/namespaces/mosaica/2008/06/webapi"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <xsd:complexType name="named_value">
       <xsd:annotation>
              <xsd:documentation>A named value.</xsd:documentation>
       </xsd:annotation>
       <xsd:sequence>
              <xsd:element name="name" type="xsd:string"
                     minOccurs="1" maxOccurs="1">
                     <xsd:annotation>
                            <xsd:documentation>Name of the
value.</xsd:documentation>
                     </xsd:annotation>
              </xsd:element>
              <xsd:element name="value" type="xsd:string"
                     minOccurs="1" maxOccurs="1">
                     <xsd:annotation>
                            <xsd:documentation>The value
itself.</xsd:documentation>
                     </xsd:annotation>
              </xsd:element>
              </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="named_value_list">
       <xsd:annotation>
              <xsd:documentation>A list of named values.</xsd:documentation>
       </xsd:annotation>
       <xsd:sequence>
              <xsd:element name="namedvalue" type="tns:named_value"
                     minOccurs="0" maxOccurs="unbounded">
                     <xsd:annotation>
                            <xsd:documentation>A single named
value.</xsd:documentation>
                     </xsd:annotation>
              </xsd:element>
       </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="call_data">
       <xsd:annotation>
              <xsd:documentation>Data required for invocation of a web API
```

```
call.
            </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
            <xsd:element name="method" type="xsd:string"
                  minOccurs="1" maxOccurs="1">
                  <xsd:annotation>
                        <xsd:documentation>Name of the method to call.
                        </xsd:documentation>
                  </xsd:annotation>
            </xsd:element>
            <xsd:element name="parameters" type="tns:named_value_list"
                  minOccurs="0" maxOccurs="1">
                  <xsd:annotation>
                        <xsd:documentation>A list of parameters (passed as
named values).
                        </xsd:documentation>
                  </xsd:annotation>
            </xsd:element>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="call" type="tns:call_data">
      <xsd:annotation>
            <xsd:documentation>Root element of the XML representing
            a MOSAICA Web API call.</xsd:documentation>
      </xsd:annotation>
   </xsd:element>

   <xsd:element name="return" type="tns:named_value_list">
      <xsd:annotation>
            <xsd:documentation>Root element of the XML representing
            a set of return values resulting from a MOSAICA Web API call.
            </xsd:documentation>
      </xsd:annotation>
   </xsd:element>

</xsd:schema>
```

Table 5 MOSAICA application web API XSD

9. APPENDIX D: MOSAICA APPLICATION PROPERTIES FILE

```
# MOSAICA webapp server settings
# should be pretty self-explanatory
# keep in sync with openfire and mysql config

# XMPP settings
si.xlab.mosaica.xmpp.server=192.168.0.134
si.xlab.mosaica.xmpp.port=5222
si.xlab.mosaica.xmpp.conf.room=mosaica
si.xlab.mosaica.xmpp.conf.host=conference

# DB settings
si.xlab.mosaica.db.name=mosaicavirtualexpeditions
si.xlab.mosaica.db.server=192.168.0.134
si.xlab.mosaica.db.port=3306
si.xlab.mosaica.db.user=root
si.xlab.mosaica.db.pass=mosaica

# conversion settings
si.xlab.mosaica.media.mencoder=/usr/bin/mencoder
si.xlab.mosaica.media.maxconv=2
si.xlab.mosaica.media.tmpdir=/tmp
si.xlab.mosaica.media.videodir=/home/jaka/dev/mosaica/videos
si.xlab.mosaica.media.maxupload=100000000
```

Table 6 MOSAICA application properties file

10. APPENDIX E: MOSAICA DB CREATION SCHEMA

```
create database mosaicavirtualexpeditions;

USE mosaicavirtualexpeditions;
CREATE TABLE vemedia (
       `ID` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
       `mediadesc` TEXT NOT NULL,
       `URL` TEXT,
       `mediafilename` TEXT NOT NULL,
       `cdate` DATE NOT NULL,
      PRIMARY KEY (`ID`)
) ENGINE=MyISAM;
CREATE TABLE vedescription (
       `ID` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
       `username` VARCHAR(45) NOT NULL,
       `vedesc` TEXT NOT NULL,
       `vexml` TEXT NOT NULL,
       `cdate` DATE NOT NULL,
      PRIMARY KEY (`ID`),
      FULLTEXT (`vedesc`)
) ENGINE=MyISAM;
CREATE TABLE vegrades (
  `ID` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `veID` BIGINT UNSIGNED NOT NULL,
  `grade` TINYINT NOT NULL DEFAULT 0,
  PRIMARY KEY (`ID`),
  CONSTRAINT `FK_vegrades_veID` FOREIGN KEY `FK_vegrades_veID` (`veID`)
    REFERENCES `vedescription` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE=MyISAM;
CREATE TABLE vecomments (
  `ID` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `veID` BIGINT UNSIGNED NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  `comment` TEXT NOT NULL,
  `cdate` DATE NOT NULL,
  PRIMARY KEY (`ID`),
  CONSTRAINT `FK_vecomments_veID` FOREIGN KEY `FK_vecomments_veID` (`veID`)
    REFERENCES `vedescription` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE=MyISAM;
GRANT ALL ON mosaicavirtualexpeditions.* TO 'mosaica'@'%';
FLUSH PRIVILEGES;
```

Table 7 MySQL script for the MOSAICA server database