# A Web-Based, Interactive Annotation Editor for the eCampus Development Environment for SCORM Compliant E-Learning Modules

Benedikt Deicke, Jan-Torsten Milde, and Hans-Martin Pohl

University of Applied Sciences Fulda
Competence Center Human Computer Interaction
hans-martin.pohl@verw.hs-fulda.de, benedikt@synatic.net,
jan-torsten.milde@informatik.hs-fulda.de

**Abstract.** The eCampus development environment was created in an interdisciplinary project at the University of Applied Sciences Fulda. Today it is a fully webbased application for the easy creation of E-Learning modules complying the SCORM standard. The webbased, interactive annotation editor for the eCampus development environment is used to both automatically and manually annotate existing OpenOffice documents in order to transform them into E-Learning modules. The editor is build using Open Source software and frameworks such as Ruby on Rails.

**Keywords:** E-Learning, Web, SCORM, eCampus, OpenOffice, Ruby, Ruby on Rails, JavaScript, user friendly, annotation, transformation.

## 1 Introduction

The creation of standard compliant E-Learning modules is a complex task. In order to simplify the process, the University of Applied Sciences Fulda created the eCampus development environment [5]. It is based on OpenDocument files which are transformed into SCORM [1] compliant E-Learning modules using XSL Transformations. The transformation relies on annotations added to the OpenDocument file using OpenOffice.org [3]. Nevertheless, the handson experience showed that the annotation process using OpenDocument styles within OpenOffice.org is still too complicated for most of the users. With the "eCampus" development environment now being a web-based and integrated system for the generation of SCORM compliant E-Learning modules, the creation of an interactive, web-based annotation editor is the next logical step. To simplify the annotation process even more an automatic analysis and annotation process is implemented, too.

## 2 The eCampus Project

The research focuses on the development of a user-friendly system for writing ELearning units, allowing authors to concentrate on the content and the didactic concept of the unit, instead of worrying about the underlying technology. It is part of the eCampus project at the University of Applied Sciences Fulda. In this interdisciplinary project, content is being created for the faculties Nutritional Sciences, Food and Consumer Sciences, Applied Computer Science, Nursing & Health Care as well as Food Technology. Within the first years of the project, many E-Learning modules were produced. All modules address prominent introductory courses of the particular program of study. This was only possible by using the eCampus framework.

The central target of the project is the implementation of a tool that decreases the complexity of the process of learning module production. The central objectives of the learning module production are an increase in the interactivity of the course, as well as an increase in the quality and quantity of self learning.

## 3 The Tools

The annotation editor is based on the existing eCampus web application which is built using Ruby [8] and the Ruby on Rails framework [7]. It is running as an E-Service on a centralized server. In order to use it, the author has to create a user account. This is required to protect uploaded documents and generated E-Learning modules from unauthorized access.

The user interface is built using XHTML and CSS. It is enhanced by JavaScript utilizing the Prototype and Script.aculo.us frameworks [4].

## 4 The Process

Using the webbased annotation editor the process of creating E-Learning modules from OpenDocument files is condensed into five simple steps: Uploading of the unannotated OpenDocument file, automatic analysis and annotation of the file, manual annotation and correction of the resulting document, transformation into a SCORM compliant E-Learning module, downloading of the finished module.

### 4.1 Reading and Modifying OpenDocument Files

In order to read and manipulate the OpenDocument a library has been developed. It encapsulates the required steps within a simple API. In order to read the contents of the document it is extracted using RubyZIP. Afterwards the content. xml file, which contains the actual content, and the *styles.xml* file are parsed using Ruby's built-in XML library [6]. Utilizing Ruby's dynamic nature, the library creates a class for every node type contained in the document's content and instantiates it as needed. This provides the possibility to dynamically overwrite the default behavior for specific node types.

### 4.2 Automatic Analysis and Annotation

The process of the automatic analysis and annotation is separated into different annotation engines. The engines analyze every stylable element in the OpenDocument's content structure using different approaches. If the analysis step is not successful, the next annotation engine tries to analyze the current element. This is continued until no annotation engines are left or an approach is successful and annotates the element. This process is repeated for every annotated element of the input text. Figure 1 visualizes this concept. Currently five annotation engines are implemented: *Keep, Tags, Styles, Bayes* and *Default.* The Keep engine stops the analysis chain if the element is already annotated with eCampus styles by checking each elements formatting style. The Tags engine analyzes the type of the current element and its context and tries to map this onto eCampus styles. For example text:h nodes with an outline level of 1 are annotated with *EC_*Modul, nodes with outline level 2 as EC_*Session*, etc. The Styles engine simply translates default OpenDocument styles into eCampus styles where possible, such as *Text body to EC_Text*. The Bayes engine tries to analyze the current elements style properties in order to detect elements like headlines or important terms which were just changed visually, but not by using using built-in OpenDocument styles. In order to achieve this, the Bayes engine is trained with correctly annotated documents. Based on the annotation and the visual style, the engine learns possible visual variations. While visiting each node in the document the Bayes engine builds "sentences" from the nodes style attributes. This results in sentences like "12pt bold italic" which are handed to a bayes classifier that returns a corresponding eCampus style or unknown if no style could be identified.

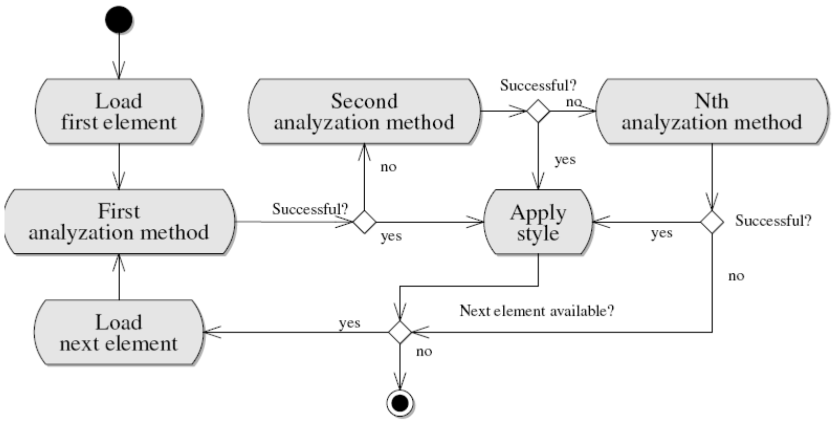


**Fig. 1.** The automatic annotation process

As well as the final transformation into a SCORM compliant E-Learning module, the automatic analysis and annotation takes several seconds (or even minutes, depending on the documents size) to finish. Therefore it needs to run asynchronously in the background. This is done marking the documents as *waiting for analysis* in the database. See Figure 2 for a complete overview of possible states. A separate processes, triggered by CRON, checks for waiting documents and runs the analysis on them. On the client side the user can see the status of the transformation. It is automatically updated using a polling AJAX request.

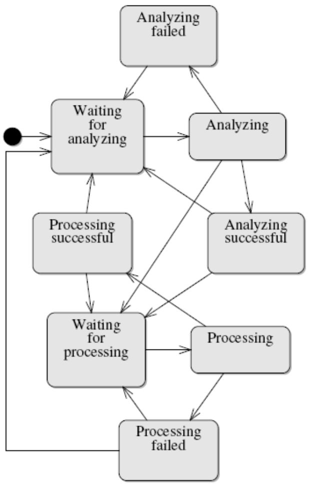


**Fig. 2.** Possible states of a document

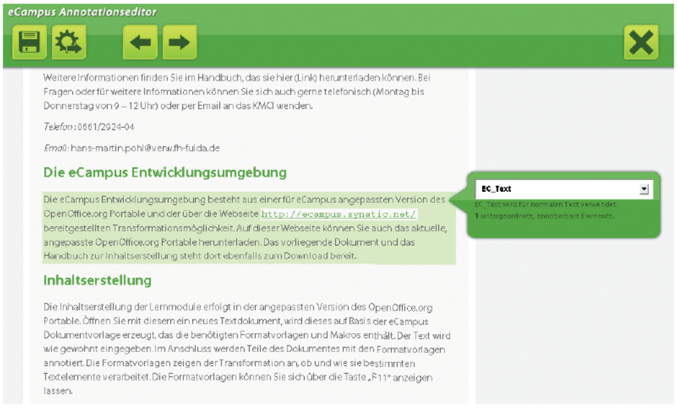
### 4.3 The Interactive, Web-Based Annotation Editor

The front end of the annotation editor for manual annotation is implemented on the client side using JavaScript. It uses Ajax techniques [2] to save changes to the document on the server. The GUI is designed to be simple and clean and focusses on the important functions needed to annotate the document. Figure 3 shows a screenshot of the final GUI. It consists of a toolbar on top, which includes buttons for navigating through the document and buttons for saving and exiting the editor. The tooltip on the right is used to actually annotate the currently highlighted element. The user is able to select the desired style by selecting it from the select-box.

The JavaScript implementation is designed to be as modular as possible. It utilizes the observer pattern to separate the various GUI components from the underlying logic. On startup of the annotation editor, every component registers itself with the *AnnotationEditor* core class. Based on the users interaction with the GUI the elements fire events on the core class, which executes the requested functionality and fires events on all registered components as a result. This enables the possibility of additional annotation methods, other than the tooltip described above. Imaginable annotation methods are: dragging and dropping the styles on the elements or a fill format mode like known from OpenOffice or Microsoft Word.

To display the OpenOffice document it is read on the server and transformed into a simple HTML representation. Each HTML elements relation to a node in the original OpenOffice document is described by adding an attribute containing the nodes XPath to the element. The nodes annotation (such as EC_Definition, EC_Section, EC_Image, etc.) as well as its family (Paragraph, Text or Graphic) are represented using CSS class attributes. Additional every HTML element that can be modified later the CSS class *annotatable* is added.

During the manual annotation the underlying CSS class attributes identifying the annotation are added, modified or removed. On modification of an element it gets marked as changed. When the user decides to save the changes, all marked elements are collected and serialized into JSON before they are transfered to the server using an Ajax request. On the server the JSON is deserialized and the changes are applied to the OpenOffice document using the library described above.


eCampus Annotationseditor

Weitere Informationen finden Sie im Handbuch, das sie hier (Link) herunterladen können. Bei Fragen oder für weitere Informationen können Sie sich auch gerne telefonisch (Montag bis Donnerstag von 9 – 12 Uhr) oder per Email an das KMCI wenden.

*Telefon*: 0661/2924-04

*Email*: hans-martin.pohl@verw.fh-fulda.de

Die eCampus Entwicklungsumgebung

Die eCampus Entwicklungsumgebung besteht aus einer für eCampus angepassten Version des OpenOffice.org Portable und der über die Webseite http://ecampus.synatic.net/ bereitgestellten Transformationsmöglichkeit. Auf dieser Webseite können Sie auch das aktuelle, angepasste OpenOffice.org Portable herunterladen. Das vorliegende Dokument und das Handbuch zur Inhaltserstellung steht dort ebenfalls zum Download bereit.

Inhaltserstellung

Die Inhaltserstellung der Lernmodule erfolgt in der angepassten Version des OpenOffice.org Portable. Öffnen Sie mit diesem ein neues Textdokument, wird dieses auf Basis der eCampus Dokumentvorlage erzeugt, das die benötigten Formatvorlagen und Makros enthält. Der Text wird wie gewohnt eingegeben. Im Anschluss werden Teile des Dokumentes mit den Formatvorlagen annotiert. Die Formatvorlagen zeigen der Transformation an, ob und wie sie bestimmten Textelemente verarbeitet. Die Formatvorlagen können Sie sich über die Taste „F11" anzeigen lassen.

EC_Text

EC_Text wird für normalen Text verwendet.
1 untergeordnete, annotierbare Elemente.

**Fig. 3.** Screenshot of the Graphical User Interface

## 5 Results and Visions

The combination of a simple user interface for the annotation and the automatic analysis and annotation process the creation of standard compliant E-Learning modules out of existing content simplifies the creation of standard compliant E-Learning modules. Being integrated into the eCampus web application the annotation editor inherits useful features like easy updating and extending. The annotation editor completes eCampus to a full web based development environment for SCORM compliant E-Learning modules.

Future development could include the possibility to rearrange elements within the document or even change the content. Additionally the current user interface needs to be fully evaluated during end-user tests. As described above other annotation methods could be implemented as well. Furthermore the automatic annotation engines offer a broad scope for new additions.

## References

1. ADL. Scorm, `http://www.adlnet.gov/scorm/`
2. Garret, J.J.: Ajax: A new approach to web applications (August 5, 2008), `http://www.adaptivepath.com/ideas/essays/archives/000385.php`
3. OpenOffice.org. The free and open productivity suite, `http://www.openoffice.org/`
4. Porteneuve, C.: Prototype and script.aculo.us. The Pragmatic Programmers, 1st edn. (2007)

5. Pohl, H.-M., Tulinska, P., Milde, J.-T.: Efficient creation of multi media eLearning modules. In: Smith, M.J., Salvendy, G. (eds.) HCII 2007. LNCS, vol. 4558, pp. 457–465. Springer, Heidelberg (2007)
6. REXML. Ruby standard library documentation (August 18, 2008), `http://www.rubydoc.org/stdlib/libdoc/rexml/rdoc/index.html`
7. Thomas, D., Heinemeier-Hannson, D.: Agile Web Development With Ruby On Rails, 1st edn. The Facets Of Ruby Series. The Pragmatic Programmers (2005)
8. Thomas, D.: Programming Ruby - The Pragmatic Programmers' Guide, 2nd edn. The Pragmatic Programmers (2005)