A LARGE AMOUNT OF FINAL PROJECTS EFFECTIVELY PROCESSED WITH MINIMAL SOFWARE REQUIREMENTS

Open Source and Platform Independent Solution: A Case Study

Michal Valenta
Czech Technical University, Faculty of Electrical Engineering, Czech Republic

Keywords: Final project Grading, XML, XSLT, HTML.

Abstract: In this paper, we present a case study based on our two-years experience with a processing of a large amount of final projects (1200 students) in a basic course on database systems (Databases 101). Each student is required to present his/her own project documentation with a relatively well-defined structure and content. The present paper offers a solution to the following bias: each student has his/her own preferred/disfavored editor and, at the same time, the teacher needs to see all the final projects in a unitary format to make the results controlable. Furthermore, we address the issue of automated (or semi-automated) processing of some parts of the controling process. Importantly, the whole process involves exclusively standardized technologies implemented in many (also open-source) tools, XML and XSLT standardized by W3C in particular.

1 INTRODUCTION AND MOTIVATIONS

There are many sophisticated tools for teaching support, however, these tools often offer only proprietary solutions (moreover, usually to problems one is not concerned with at the moment). In this paper, we show how to use very common and basic tools to achieve a relatively easy control (and feedback options) for the teacher on one hand and a relatively non-constrained requirement on the students. Hence, a win-win solution with minimal technology requirements.

The problem we deal with is this: there is about 600 students per semester at the basic Database systems course (Databases 101). Each of the students has to present a final project. This project has a form of a paper that has to be processed and controlled by a teacher. Hence, the challenge is to make it both possible for the teacher to control the papers relatively easily on one hand and, on the other, not to impose a system on a student (in particular a system he or she is not used to work with).

There is a plenty of systems available for such purposes (we have some experience with moodle on CTU).
Starting with open-source solutions, we are usually faced with a poor documentation. Moreover, we have to invest a lot of effort to install (and maintain) these tools. Adding furthermore long and expensive training period both for the teacher and the student, we want something simpler for our purposes.

Additionally, it is very often the case that these systems are equipped with too many functions that are usually unnecessary for both the teacher and the student.

So, it seems that we want to balance the overwhelming power of these systems with our real needs of efficient and relatively easy control of the final projects.

In this paper, then, we present the initial problem (a large amount of final projects that need to be controlled; the requirements will be defined below) and our solution to this problem based on the following technologies - http, webdav, xml and xslt. The solution has gone through a two year cycle, hence, about 1200 students participated in the course Databases 101.

The main theses of this case study are:

- basic course of DBS (Databases 101),
- several hundreds of students per semester,
- individual projects,
- certain requirements for both the structure and the final format,
- crucial impossibility to impose one or few editors on the students,

Valenta M. (2009).
A LARGE AMOUNT OF FINAL PROJECTS EFFECTIVELY PROCESSED WITH MINIMAL SOFWARE REQUIREMENTS - Open Source and Platform Independent Solution: A Case Study.
In *Proceedings of the First International Conference on Computer Supported Education*, pages 235-240
DOI: 10.5220/0001972702350240

- further advantage: possible automated processing of (at least) some parts of the final projects.

The rest of the paper is organized as follows: firsts, we give a detailed description of the requirements on the final projects from DBS (Databases 101) in the section 2. Advantages of the implementation of our solution with respect to the technologies used and their compatibility as well as students' point of view is presented in the section 3. In the section 4, we present our experience after two semesters[1]. We consider two groups of students here – students familiar with XML/XSLT technologies vs. those without such experience. To conclude, in the section 5, we summarize our findings with respect to the above specified statements. Also, we shortly discuss the future work and potential extensions here.

2 FINAL PROJECT REQUIREMENTS

Our aim is to provide a structure for the students' final projects. This structure must be flexible enough to reflect the needs of the descriptive parts of the document. On the other hand, it must be rather strict, as it defines the names and the order of the required parts of the document. The requirements for the final project structure are not so strict to allow fully automated processing as it is usual in programming courses, see (Cheang et al., 2003) and (Helmick, 2007) for example.

XML (Extended Markup Language) seems to be an ideal format for these purposes. So, the final project is ought to be presented as an *XML document*. There are some basic rules on syntax of a XML document. A document which satisfies such rules is called *well-formed*. See W3C pages for detail (`http://www.w3c.org`).

However, a well-formed XML document still does not meet our expectation on the final project structure. We need to strictly specify the names of individual final project parts (*element*s in XML terminology), the order of these elements, and their cardinality (the number of their occurrences in certain context).

For example, an element `description` which represents one part of the final project document, consists of exactly one element `title` followed by at least one (but potentially more) paragraphs. Each paragraph consists of an element `para` optionally followed by an element `comment`. In DTD formalism (see bellow), this requirement looks as follows:

```
description (title, (para, comment?)+)
```

[1]Databases 101 runs only in the Fall semester.

For these purposes, a *document type description* can be specified. A document is then checked against this structural description. If it satisfies all the requirements on the document type description, it is marked as *valid*.

There are three notations/languages used for a document type definition – DTD, XML Scheme, and RelaxNG. We choose DTD, since it is the simplest one, yet, it is powerful enough for our purposes. Each DTD specification can be automatically translated into XML Scheme as well as RelaxNG specification. These facts, in turn, are important for XML document editing. Some editors do not support DTD, but require XML Scheme, for example.

A complete DTD specification for a DBS final project is included in Appendix A.

2.1 Final Project Parts

Let us shortly describe the main parts of the final project. We will refer to them in chapter 3.

2.1.1 Domain Description

First, students are required to specify their own database for an implementation. They have to describe the context they are going to model on a conceptual level and, subsequently, they have to implement it in SQL server[2]. This part consists of minimally one, typically several paragraphs.

2.1.2 Conceptual Model

As a second step, students formalize their domain in ER (Entity Relationship) notation. They use a special tool for ER modeling in this stage of the work. They are required to include a figure of their ER model and an URL link to the source code of this model into their final project documentation.

This part also requires at least one additional paragraph, in which the students discuss the potential problems of loops that might arise in their model and they also formulate additional integrity constraints (such that can't be described in ER notation).

2.1.3 Relational Database Implementation

This stage of the final project consists of several individual parts. We'll describe each of them in the following paragraphs.

[2]As a rule, Oracle is used for all the courses.

SQL Script for Database Creation. It can be created automatically from the tool used for the ER model creation. Students can edit and comment on this script. The SQL script is then included as a URL link into the final project documentation.

SQL Script for Database Initialization. Students are required to create a set of INSERT statements in order to fill their database. It is necessary for further SQL statements development. It is included as a URL link in the final project documentation.

SQL Statements. Students have to design at least 25 SQL statements of various complexity over their database. Various types of SQL statements are described in a table, which is then included in this part of the final project documentation. Students have to cover each column of the table by at least one SQL statement. Each SQL statement consists of a description, SQL code and potentially a formulation of a query in relational algebra notation (students have to provide a relational algebra notation for at least 10 queries).

SQL Script with SQL Statements. In order to control for a correctness of their SQL code, students are required to attach a sql script consisting of only SQL statements. This script can be created automatically from the final project documentation using another XSLT transformation; see section 3 and figure 1 for details. The script is included as a URL link in the final project documentation.

SQL Statements Report. In order to show that their SQL statements really work, students have to include a report with SQL server responses to their SQL statements. Again, this document can be created automatically running previous sql script in SQL server. See figure 1 for details. As before, this report is included as a URL link in the final project documentation.

2.1.4 Conclusion

This section serves exclusively for the students' review of their experience. It thus has at least one paragraph in the Final project documentation.

2.2 Comments

We decided to add (optional) comments to most parts of the final project docementation structure. It is meant as a place for discussion or comments on a particular solution both for the students and the teacher. It is implemented using a button with java script functionality which allows to shows or hide all comments in the final work in its web presentation. See chapter 3 for details.

3 IMPLEMENTATION

3.1 Technical Description

Figure 1 provides a brief description of the technologies used in the final project and of the final project document processing. Let us describe the processes in detail here. We will refer to the labels of arrows in figure during our explanation.

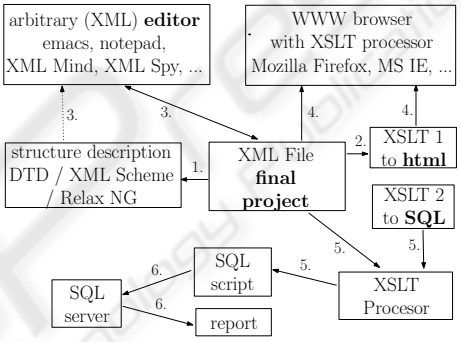

Figure 1: Tools envolved in the final work processing.

1. Final project document points to a related DTD in a meta-element `<!DOCTYPE>`. Hence, each tool which is used for a final project document can access and use a DTD specification to validate the document.
2. In another meta-element – `<?xml-stylesheet>`, the final project document points to a XSLT1 transformation program (by URL). It transforms the final project document into html. Visual formatting of output html document is done through CSS (Cascade Style Sheet) which is included in the head of the output html document by standard html tag `<link>`.
3. Editing of the final project document. This is the main task in the whole process. A document can be edited either by a student (most often), potentially by a teacher (especially comment elements). Editing can be done either locally on students'/teachers' PC, or directly on a server (if it is supported by the access protocol). Practically, we are using WebDAV protocol which allows remote file editing. Related DTD of the document can be used by the editor (dotted arrow), but a document can be (and it often is) edited by an arbitrary (non-XML specialized) editor.

4. This arrows represents the main benefit of our solutions. The final document is placed on a web server into a folder accessible through http protocol. If a user uses WWW browser with XSLT processor (actually, this feature is implemented in many browsers: Mozilla, Firefox, Opera, Safari, Microsoft IE), then the XML document is automatically transformed via related XSLT (see label 2. above) and displayed as html page. Unfortunately, the relation to XSLT transformation inside XML document (although it is standardized!) is not implemented in a standard way (always – i.e. in MS IE). Hence, if one sustains on using MS IE for his/her final project, it is necessary to (slightly) modify XSLT 1.
5. This processing is optional. Another XSLT template (XSLT 2 in figure) can be applied to the final project document. It extract only a part related to SQL statements and it creates a SQL srcipt which can be directly run in a SQL server.
6. Actually, our XSLT 2 transformation is built for optimal use with Oracle SQL server. It includes settings for sqlplus command tool and in this way it generates a html report with responses to SQL statements. It is also a required part of the final project (see 2 for details).

3.2 Users' Point of View

In this subsection, we're going to describe the whole process of the creation of the final project from the point of view of a user – i.e. a student as a creator and a teacher as a reviewer. We'll do it in a way of "use cases".

Suppose now that the final project files are placed on a server, which provides http and WebDAV (possibly ftp or scp) access to files. Both the student and the teacher know URL (Unified Resource Locator) which points to a particular final project documentation.

3.2.1 Student

Edit Final Project. The student can edit the final project document using an arbitrary editor. As before, it can be done locally by transferring the document, editing it locally and then posting it back to the server. And again, it can be done by a special feature of a remote editing as well.

View Final Project. In a web browser with XSLT support (Firefox is recommended) type URL of XML document. It is displayed with a proper formatting. Use embedded push button to hide/show comments in the document.

Deliver Required File to the Server. An arbitrary file can be posted to the project folder on the server and then linked from the final project document via URL. The delivery is done via server supported technology (WebDAV in our case). An arbitrary tool can be used. WebDAV protocol is also supported in tools like Konqerror in Linux KDE environment or Explorer in MS Windows.

Deliver Figure to the Server. A figure of conceptual model is required as a part of the final project. It can be delivered in the same way as another files. See the paragraph above.

Create Script with SQL Statements. Use any xslt procesor and apply XSLT2 template to the final project document. There are free as well as payed xslt prcoessors (Xerces, xsltproc, or xalan for example). Moreover, XSLT2 template can be downloaded from the project server.

Run SQL Script and Obtain Report. In sqplus run sqlscript which is the output of XSLT2 transformation. It creates the report automatically.

3.2.2 Teacher

View Final Project. In a web browser with XSLT support (Firefox is recommended) type URL of XML document. It is displayed with a proper formatting. Use embedded push button to hide/show comments in the document.

Comment on a Part of the Final Project. Using an arbitrary editor, edit a chosen comment element (it can also be added according to DTD specification). It can be done locally by transferring the document, editing it locally and then posting it back to the server, or, as before, by using a special feature of remote editing.

3.2.3 Recommended Editor

Let us note that any kind of an editor (either XML or non-XML one) can be used, but there is also a recommended (i.e. supported) one.
We choose *XMLMind* editor. It is implemented in java language, so it is a platform independent. A community edition is free of charge. On the other hand, a payed version has the posibility to edit files directly on a remote server via WebDAV protocol (which has proven as beneficial in our case).
The editor works in WYSIWYG (What You See Is

What You Get) mode. We spent some effort to provide formatting identical to the final presentation via html (see figure 1 and description of label 4., as discussed in chapter 3).

A support of our final project document processing is implemented as a standard XMLMind plugin. So, its installation and configuration is trivial and user-friendly.

4 OUR EXPERIENCE

The students in the DBS course (Databases 101) are not a homogenous group of students. In particular, there are students who have already passed a course on XML Technologies (hence, they are familiar with XSLT transformations). On the other hand, there are students without a prior knowledge of the XML technologies. Nevertheless, it turned out that the prior knowledge of the XML technologies is not a special advantage, as we might have suspected from the beginning, for the Databases 101 course. That is, all the students are capable to successfully submit their final projects and, moreover, the students found the process described here very helpful and useful. In particular, they are thrilled by the fact that the rigid structure of the documentation (via its *validity*) contains all the required parts and they do not have to concentrate on the formatting side of the final project. Interestingly, vast majority of the students used the simplest possible editor (Notepad) for editing the XML. However, it seems that the possibility to write comments (as described in chapter 2) is not widely used neither by the students, nor by the teacher.

5 CONCLUSIONS AND FUTURE WORK

Two years after having implemented the proposed solution we can summarize that the project was very successful and useful not only for the students, but – importantly – also for the teacher responsible for the control of the final projects. The proposed solution meets the requirements we diagnosed in the first chapter: it is simple, it is not dependent on a proprietary solution, it is not time consuming, it is based on an open source and standardized technologies.

In the future, we would like to extend the automatic SQL control of the final projects. Once a script with SQL questions is automated, it will be possible to run it on SQL server and automatically analyze the report. However, the possibility of further automated control of such final projects is directly proportional on the structure and requirements of the final project. Still, leaving aside these potential extension, we found the proposed way very prosperous and worth further exploration.

The overview of the proposed solution can be found at
`http://service.felk.cvut.cz/courses/XE36DBS/xml/test/test.xml`
for now only in Czech. However, for the camera ready paper, the English version should be available, including instructions the student were given and the comments on the technologies used. Mozilla or Firefox are recommended.

ACKNOWLEDGEMENTS

I would like to thank all the students who participated in Databases 101 for their cooperation and comments.

This work has been supported by Ministery of Education, Youth, and Sports under research program MSM 6840770014.

REFERENCES

Cheang, B., Kurnia, A., Lim, A., and Oon, W.-C. (2003). On automated grading of programming assignments in an academic institution. *Comput. Educ.*, 41(2):121–131.

Helmick, M. T. (2007). Interface-based programming assignments and automatic grading of java programs. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 63–67, New York, NY, USA. ACM.

APPENDIX A

Complete DTD for Final Work From DBS

Explanation:

`#PCDATA`	stands for an arbitrary string (text)
`comment?`	zero or one occurence of element `comment`
`comment+`	one or many occurences of element `comment`
`comment*`	zero, one or many occurences of element `comment`

DTD specification link (URL) can be included in each XML document, so validating software can find and access it.

```
<!ELEMENT sproject (course, author, content)>
<!ELEMENT course (cname, code, semester,
      seminar)>
<!ELEMENT author (aname, uname, email)>
<!ELEMENT content (declaration, title,
      description, data_model, queries,
      scripts, conclussions,  references)>
<!ELEMENT cname (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT semester (#PCDATA)>
<!ELEMENT seminar (#PCDATA)>
<!ELEMENT aname (#PCDATA)>
<!ELEMENT uname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT declaration (#PCDATA)>
<!ELEMENT description (title, (para, comment?)+)>
<!ELEMENT data_model (title, dm_picture, comment?,
      dm_discussion)>
<!ELEMENT scripts (title, (para, comment?)+)>
<!ELEMENT queries (title, dotazy, comment?,
      pokryti_dotazu, comment?)>
<!ELEMENT conclussions (title, (para, comment?)+)>
<!ELEMENT references (title, (para, comment?)+)>

<!-- paragraph -->
<!ELEMENT para (#PCDATA | link)*>

<!-- comment -->
<!ELEMENT comment (para)*>

<!-- dm_picture -->
<!ELEMENT dm_picture (mediaobject)>

<!-- dm_discussion -->
<!ELEMENT dm_discussion (para, comment?)+>

<!-- table -->
<!ELEMENT pokryti_dotazu (kategorie_dotazu+)>

<!-- kategorie dotazu / query category-->
<!ELEMENT kategorie_dotazu (entry+)>

<!-- entry -->
<!ELEMENT entry (#PCDATA )>

<!-- link -->
<!ELEMENT link (#PCDATA)>
<!ATTLIST link
      url CDATA #REQUIRED
      >

<!-- dotazy / queries -->
<!ELEMENT dotazy (dotaz)+>

<!-- dotaz / query -->
<!ELEMENT dotaz (popis_dotazu, comment?,
      relacni_algebra?, comment?,
      (sql, comment?)+)>

<!-- popis dotazu / query description-->
<!ELEMENT popis_dotazu (para)+>

<!-- rel. algebra / relational algebra -->
<!ELEMENT relacni_algebra (#PCDATA)>

<!-- sql -->
<!ELEMENT sql (#PCDATA)>

<!-- mediaobject -->
<!ELEMENT mediaobject (imageobject)>

<!-- imageobject -->
<!ELEMENT imageobject (imagedata)>

<!-- imagedata -->
<!ELEMENT imagedata EMPTY>
<!ATTLIST imagedata
  fileref  CDATA   #REQUIRED>
```