We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS
BOOK CITATION INDEX
INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

XML Data Access via MOODLE Platform

Aleksandra Werner and Katarzyna Harężlak
Silesian University of Technology,
Poland

1. Introduction

Nowadays, exchanging information plays a crucial role in both human communication and business tasks realization. More and more often, this process is performed with usage of computer systems. However, their variety, resulting from differences in time of their creation, exploited tools and platforms, enforces a necessity to their integration to achieve effective co-operation of business entities. Unfortunately, such computer systems contain data in incompatible formats and large amounts of information must be converted. Moreover, in the newly created software, an idea of outsourcing of functionality can be observed, what entails a need for an access to the same data by different applications. Cooperation of sales systems with one invoice system is a great example of such situation.

The aforementioned systems collaboration requires standardization of the exchanged data. The XML language is one of the most popular standards used for this purpose. XML data is stored in a plain text format independent from a software and hardware, making a process of creating data that is intended to be shared by different applications, easier.

XML is the meta-language for structured data description. There are no mandatory objects defined, only a set of rules of their creation. This is a reason why there are many fields of XML usage. For example it is used in financial and economic institutions of European Union. Furthermore, text editors, like MS Word (Rice, 2006), allow for saving documents in XML format, which is utilized in many companies to write their internal documentation and in social life for exchanging documents between citizens and offices.

For these reasons, knowledge concerning rules of constructing XML documents, ability to analyse their content and ways of their storage seems to be worth acquiring.

XML document consists of data described in accordance with the defined XML format and has to meet specified conditions to be regarded as correct. XML document can be analyzed on various levels of abstraction (Bray&Paoli&Sperberg-McQueen, 1998). On the highest level, a document is a tree with at least one, root element consisting of pair of tags - opening **<tag>** and closing **</tag>.** Other elements must be located inside the root. Between tags, contents can be placed. Element can be an empty one, which is marked by one tag in form of **<empty_element/>.**

Further analysis shows that XML document can be treated as a combination of tags and text data. The task of tags is to define a structure of a document. Correctness of such document can be considered in two terms:

- document is complete enough to be interpreted by a web browser,
- document has proper syntax structure.

Creating XML document can be realized with usage of any text editor but more conveniently is to utilize tools dedicated to manage XML files. Among them, for example, there can be enumerated (O'REILLY xml.com):

- Adobe FrameMaker, available from http://www.adobe.com.
- XML Pro, available from www.vervet.com/.
- XML Writer, available from http://xmlwriter.net/.
- XML Notepad, available from http://www.microsoft.com/download/en/default.aspx.
- eNotepad, available from http://www.edisys.com/Products/eNotepad/enotepad.asp.
- XML Spy, available from http://www.altova.com/.

They differ in various features - some of them have to be paid for and others are free, some of them are difficult in use, and other are easy to exploit. But none of them allows for organizing complete teaching process. Therefore, the possibility of constructing such comprehensive procedure in the MOODLE environment was considered. The goal of the research was to provide the complete tool supporting educational process of XML data management.

Solution of this problem was carried out in accordance with the principles of the major methodologies for software development, taking into consideration the scope of the problem being solved. The particular steps of project development covered:

- formulating the goal and assumptions of the research,
- analyzing the field of research and dividing the general knowledge about XML language into groups of issues,
- specifying the desired functionality in scope of XML data management,
- developing the algorithms covering given functionality,
- selecting the technologies needed to achieve the desired results and implementing the appropriate mechanisms.

Thanks to the ability to split the issues concerning the XML language into the smaller subgroups, the principles of Agile software development (Martin, 2002) were applied during the project realization. The effects of each iteration were demonstrated to potential users and their remarks were taken into consideration in subsequent steps. Preliminary results obtained during the research confirmed initial assumptions regarding usability of the elaborated mechanisms, so their evaluation for a wide group of beneficiaries is planned as a future work.

A documentation of work progress and obtained results presented in a chapter has the following structure. The first part includes general information about the MOODLE platform and XML language (sections 1, 2). The second part, including sections 4.1 and 4.2, presents the original solutions allowing for creation and analysis of XML documents. Point 4.3 describes synonyms defining tools, enhancing flexibility of these mechanisms usage. These issues are also discussed in section 6.1, where the activity of a specially designed block structure is explained. This section provides the mechanisms for creating the individual educational path (point 5) as well.

The third section describes more advanced mechanisms for the XML activity, including both DTDs and XML Schemas (point 6.2). Besides, the mechanisms for XML usage in relational databases are described in the 7th point of the chapter.

In the conclusion, the obtained results are summarized and directions for future work are formulated.

2. The MOODLE platform

IT development, the more and more popular access to the Internet and variety of tools enabling using it in the field of teaching (Nedeva, 2005), encourage many educational institutions and companies to expand their offers with e-learning courses (Daku, 2009, Dobrzański&Brom&Brytan, 2007). Owing to that, quicker access to knowledge and learning costs reduction can be achieved. Course materials can be studied by the employees or students at a convenient time and place, which gives them a chance to equalize possibilities of getting an education.

However, when using the available e-learning tools, two problems are observable.

- So far, usage of e-learning platforms is limited to theory presentation and exchange of teaching materials, whereas, in some knowledge branches - especially in the field of technical science - teaching practical usage of theory, experience achieved in real environments and learning from one's own mistakes play an important role. Current functionality of e-learning platforms does not provide such possibilities.

- Second problem results from asynchronous type of learner-teacher communication, which makes both sides dependent on their mutual activity. Solutions of complex problems are usually more extensive than just a yes/no answer. It should be taken into consideration that very often many aspects of the solution should be analyzed, which can have a massive effect on delays in remote communication. Thus, an automatic estimation of solutions inconvenience should be introduced.

Dealing with the aforementioned problems will enrich and make teaching process more effective on various levels of education. What is more, in many cases, capabilities of tools, supporting e-learning are higher than only knowledge presentation and make elimination of described inconveniences possible.

Because, one of the most popular and free environments for e-learning is Modular Object-Oriented Dynamic Learning Environment [MOODLE] (MOODLE Home Page –Statistics, Braccini&Silvestri&Za&D'Atri, 2009), it was used in the research in order to extend it with the new mechanisms.

MOODLE course management allows teachers to build their courses in an optimal way for their learners and subject matter. The platform capabilities include many different mechanisms – i.e. activities, giving trainees possibility to test and to resolve given problems in practice and passive resources, allowing the knowledge presentation. The Open Source distributed MOODLE environment, has the specific (i.e. modular) organization. Besides, it cooperates with a database server – one of the following can be used: MySQL, PostgreSQL or MS SQL Server (MOODLE Home Page – About MOODLE). This features caused, it was very suitable tool for introducing new mechanisms to a group of already existing activities.

Due to the fact, in previous papers (Harężlak&Werner, 2009; Harężlak&Werner, 2010), the MOODLE module structure was explained in details, now only the key issues will be described.

After installing the whole environment (i.e. Apache Server, PHP and chosen database server), all MOODLE courses modules, such as: chat, forum, glossary, lesson, quiz, etc, are placed in one folder */www/mod* of WAMP server installation path[1]. Every module has its

---

[1] All information about the MOODLE software structure relates to Windows operating system.

separate folder of a corresponding name and the similar structure. The most important - and obligatory in each folder - files, are: **mod_form.php** and **view.php**. First file is responsible for a contents of page seen by a teacher during editing an activity, while second one - for contents of page seen by course participants during executing an activity. Thus, the new module implementation is in practice reduced to creating a new folder with a known structure, and to providing the desired functionality of component files by the PHP or Java Script coding.

Taking the important role of XML language in the exchange of a wide variety of data into consideration, the new custom XML mechanisms were developed in order to make the completion of educational tasks possible.

Therefore tasks (activities) with a different exercise level of difficulty were prepared in a module. Among them, two groups can be enumerated. First of them consists of simple operations on an XML file:

- creating a simple XML document,
- analysing of an XML file content,
- inserting a single row of data and loading data from an XML file.

In the second one, more complex tasks, regarding database mechanisms for XML data management, are considered:

- designing and creating database structures, dedicated to XML data collecting,
- building XML schemas and assigning them to the database structures,
- retrieving relational data as XML and querying XML data with usage of advanced methods.

The chapter will present novel mechanisms guaranteeing comprehensive, interactive XML language learning.

3. Sample XML data structure

The analysis of the proposed solutions will be conducted on the basis of the example presented below. The XML document instance that describes titles, authors and supervisors of students thesis (Master and BA) in the Polish universities, will be taken into consideration. Let's assume, the task formulated in XML module by the tutor, is:

Write the well-formed instance of the XML document for the universities students' thesis data. First, data about the name of the university should be stored. Then – subsequently – for each student after defence, the student register number, the type (Master or BA) and title of thesis, the year of defence, finally, the name of student's promoter and his titles should be placed.

Additional requirements: use the tag <List> for the root element. Don't use tag attributes - only Element components. All data about the Master or BA thesis of the specified student should be described in 1 element. The description about each kind of a student's thesis should be contained in separate elements. The solution should contain an example data for 2 students from different universities. Suppose, first student is after Master and BA defence in Silesian Technical University, while second – only after BA in AGH University.

There are a lot of possibilities to formulate the given XML document instance for the given task. Two sample alternative XML documents of given requirements, where the names of the tags are not significantly different (*University* vs. *UnivName*, *TypeBA* vs. *BA*, etc.), are shown in fig. 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<List>
  <UnivName>
    Silesian Technical University
    <student>
      1357321
      <BA>
        2008, PhD Adam Winkler
      </BA>
      <Master>
        2011, Prof. Jan Boltz
      </Master>
    </student>
  </UnivName>
  <UnivName>
    AGH University
    <student>
      1357321
      <BA>
        2008, PhD Adam Winkler
      </BA>
    </student>
  </UnivName>
</List>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<List>
  <Univeristy>
    Silesian Technical University
    <student>
      1111122
      <TypeBA>
        2005, PhD Anna Michna
      </TypeBA>
      <TypeMaster>
        2008, PhD Anna Michna
      </TypeMaster>
    </student>
  </Univeristy>
  <University>
    AGH University
    <student>
      1111133
      <TypeBA>
        2011, PhD Joe Winter
      </TypeBA>
    </student>
  </University>
</List>
```

Fig. 1. Two sample alternatives of XML document instance.

The XML Schema shown in the fig. 2 defines the structure of described sample XML document, will be used to present mechanisms introduced in the MOODLE platform.
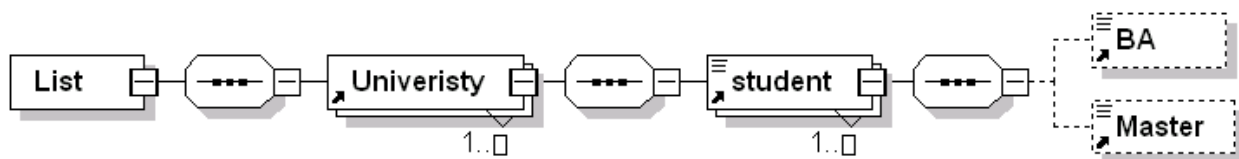
Fig. 2. The XML Schema of an example XML document.

4. Defining an XML document

As it was presented in the Introduction, XML documents can be utilized in many areas of life, but to be useful, they have to be prepared accordingly with a given structure, comprehensible for every interested part. Therefore, the most basic skills needed for the XML data management are an ability to formulate and analyze well-formed XML documents. Their teaching is supposed to be conducted in two directions. On one hand, the knowledge concerning rules of creating XML documents should be presented. On the other hand, there is a need to take care of gaining an experience in this field. Basis for obtaining such experience is an individual, practical usage of theory supported by feedback regarding types and places of made mistakes.

Due to that fact, functionality for XML document defining was designed for the MOODLE platform (Fig. 3), in which verifying of a document instance correctness was the most important problem to solve. It was accomplished with usage of both the MOODLE platform

and database objects. Two groups of tables were introduced. The first one is dedicated to the teachers and serves as a container for descriptions of proper documents structures. The second one is indented to collect course participants' solutions. Validation of this solution is performed by the comparison of the contents of both groups. The appropriate functions are responsible for realization of this process.
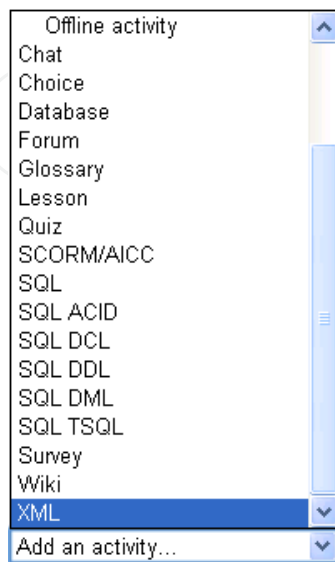

Fig. 3. The new XML activity.

4.1 Defining a structure of an XML document template

In the first research phase there was assumed that a teacher preparing tasks to be solved by course participants knows mechanism of the MOODLE platform well and does not have to know issues concerning computer science. So, mechanisms, introduced into this environment, had to be prepared in a way enabling teacher to use his/her skills.

The initial step in the research was to adjust a teacher's window to new functionality, which is XML document structure management. For this purpose, the **mod_form.php** file for a new activity, called XML, was modified (Fig. 3). Changes regarded both its construction and logic, it contained. Structure of the window was extended with a few elements. The most important one is the *htmleditor* named xml_entry (Fig. 4), which is designed to be filled
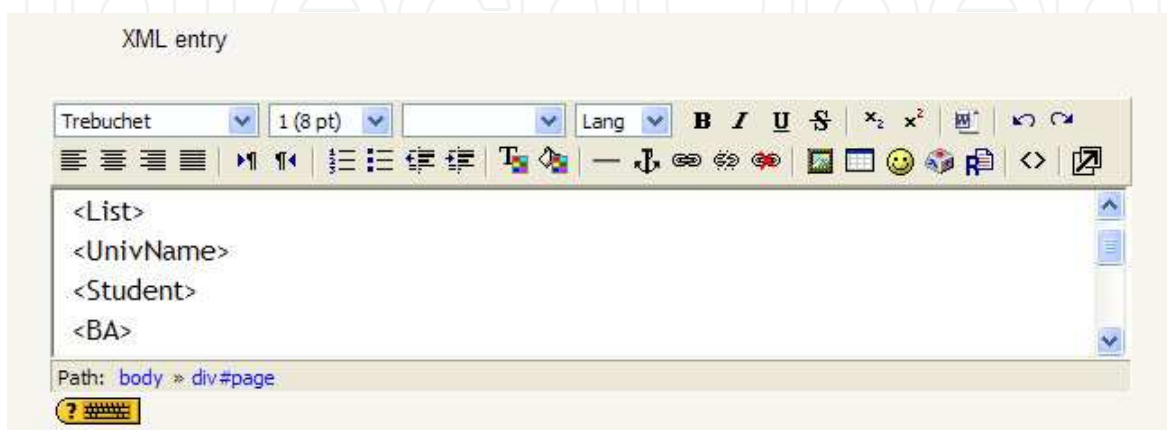
Fig. 4. The xml_entry htmleditor element visible in the teacher's window.

with an XML document template structure. Because content of this element will be different for various tasks, the identifier of a task should be entered as well (`task_number`**,** Fig. 5).

```
$mform->addElement('htmleditor','xml_entry',get_string('XML entry','xml'));
$mform->addElement('text','task_number',get_string('task number','xml'));
```
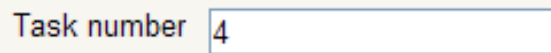
Task number 4

Fig. 5. The `task_number` text element visible in the teacher's window.

The provided XML template, for further use, has to be inserted into the database. Therefore, the *validation* function, which is run during the process of saving the newly defined activity, was equipped with required logic. Details of a database and credentials can to be provided by a teacher in next four fields (Fig. 6):

```
$mform->addElement('text','dbhost',get_string('dbhost', 'xml'));
$mform->addElement('text','dbname',get_string('dbname', 'xml'));
$mform->addElement('text','dblogin',get_string('dblogin', 'xml'));
$mform->addElement('text','dbpassword',get_string('dbpass', 'xml'));
```

or can be prepared by a module administrator with usage of a *setDefault* function, as is presented below:

```
$mform->setDefault('dbhost','moodle_server');
```

Database Settings

Database host   moodle_server
Database name   xml_managemet
Database login
Database password
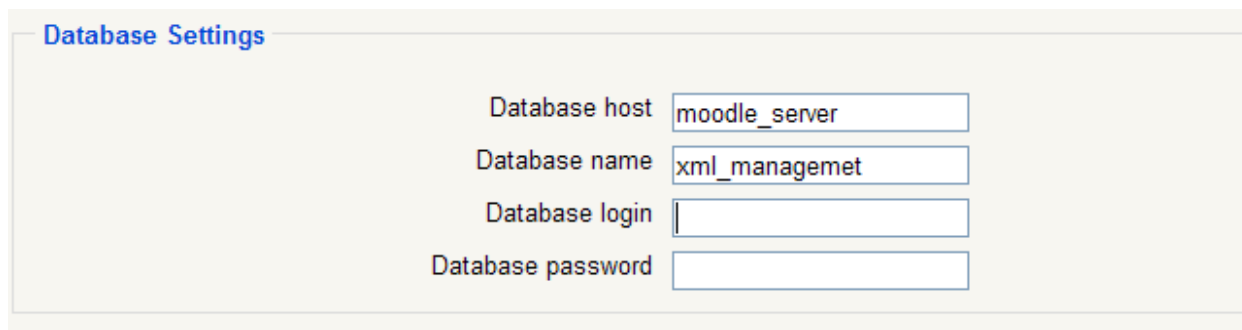
Fig. 6. Elements defining parameters for a database connection.

This database, as mentioned above, must possess a special *Tag_templates* table, in which each tag from a template structure and its level in tags tree are stored in one row. Additionally, this information is complemented by the number of the task being created. However, before it is possible, XML template must be divided into substrings - one for each tag. This operation is also performed by the logic of the *validation* function. Unfortunately, usage of *hmleditor* element entailed problems regarding recognizing limiters of tags - e.g. < and > characters. They belong to a set of special symbols of HTML language and are represented by strings in form of *<* and *>* respectively. In consequence, obtaining string representing a tag must be preceded by extracting these chars. This problem can be avoided by using *textarea* elements:

```
$mform->addElement('textarea','xml_entry',get_string('XML entry','xml'));
$mform->addElement('textarea','synonyms',get_string('Synonyms','xml'));
```

During the process of XML template transformation a level of a tag is calculated as well. For example, if the aforementioned sample structure was taken into consideration, the result of acting of the *validation* function would consist of elements presented in the Table 1.

List	1
UnivName	2
Student	3
BA	4
Master	4

Table 1. Sample result of the *validation* function acting.

The *Validation* function is also responsible for checking contents of *Tag_template* table. If rows of another template for the considered task are found, they are replaced by a newly defined set.

4.2 Validating solution correctness

When the whole process of defining XML document template is finished, new activity can be utilized by course participants in the same way as all predefined in the MOODLE platform objects and is accessible in a list of participant's tasks. Window of the XML activity includes tree elements (Fig. 7): task contents, text area for its solution and *Check structure* button, which starts a procedure of verifying a correctness of a solution.
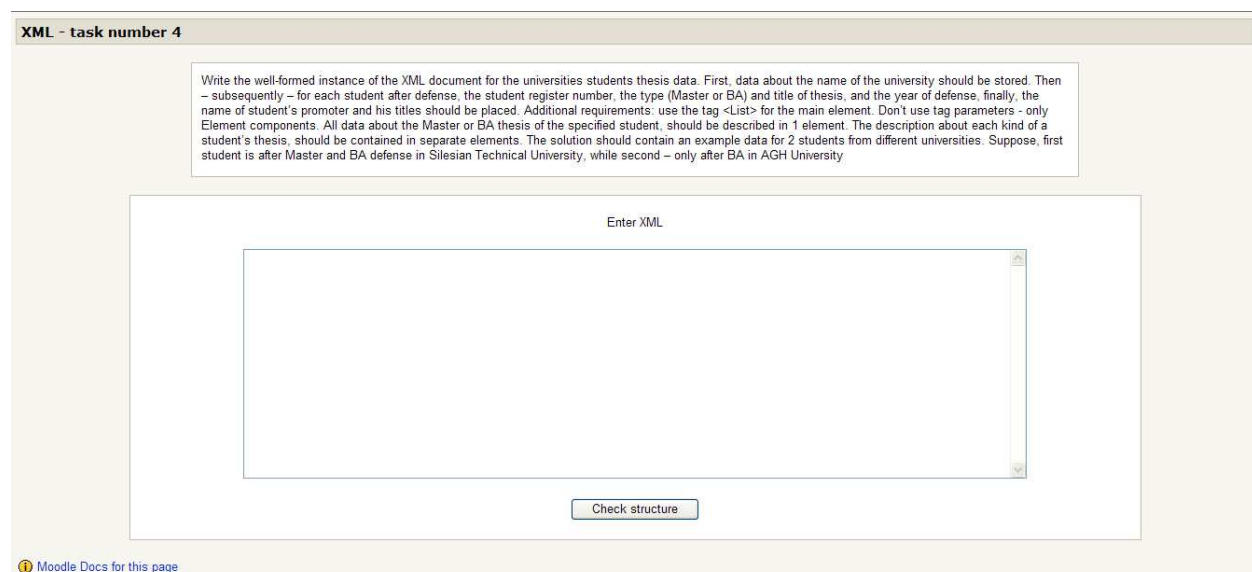
Fig. 7. View of a XML course participant's window.

This procedure, included in logic of the **view.php** file, is based on data inserted to the *Tag_templates* table and similarly, begins with splitting solution into substrings. Because a base for this partitioning is the symbol of a new line, the requirement ending each tag and value by this character has to be fulfilled. For each substring a level is calculated, which, subsequently, is compared with the level defined for a related tag in a template. In case of detecting a difference participant receives an appropriate message (Fig. 8). At the end of the

process, an existence of a corresponding opening and closing tags is checked. A proper solution is acknowledged with a suitable notification (Fig. 9).

Fig. 8. View of a participant's window with a badly prepared solution.
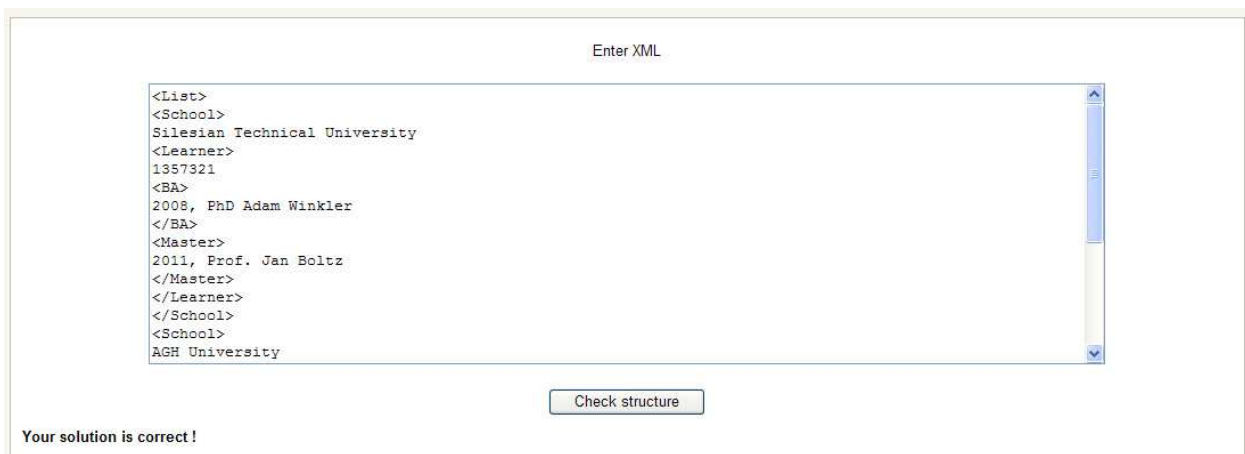

Fig. 9. View of a participant's window with a properly prepared solution.

Independently, every analyzed element is remembered in *Solutions* table. A row of this table includes task number, tag, its level in a structure defined by a participant and user identifier assigned to her/him. Owing to that, a teacher is able to analyze the result of a participant's work. Additionally, a new row, containing data on task and a participant solving it, is added to *Grades* table. If a solution is acceptable, the *passed* column is checked in this record. In other case column named *attempt* is incremented. The number of possible attempts is defined by a teacher in a task configuration window, through the additional element introduced into the **mod_form.php** file (Fig. 10):

```
$mform->addElement('text','attempts_number',get_string('attempts','xml'));
```


Fig. 10. The `attempts_number` text element visible in the teacher's window.

4.3 The synonyms

Analysis of tasks' solutions can indicate that course participant prepared a good solution of a task, but used different tags from those defined in a template. XML elements must follow a few naming rules (e.g. names cannot contain spaces), but any name can be used, because no words are reserved. Therefore mechanisms for defining synonyms were developed. Again, the **mod_form.php** file was utilized. The set of elements defining teacher's window was extended with one more html editor for synonyms providing (Fig. 11).

```
$mform->addElement('htmleditor','synonyms',get_string('synonyms', 'xml'));
```

A synonym format is specified to facilitate an analysis of entered words. Each tag with additional words should be surrounded by square brackets: "*[*" ,"*]*". First word in that scope should be a tag ended by ":" character. After that, synonyms for a given tag, separated by a space, could be placed. Once more, example of the universities students' thesis data will be taken into consideration.
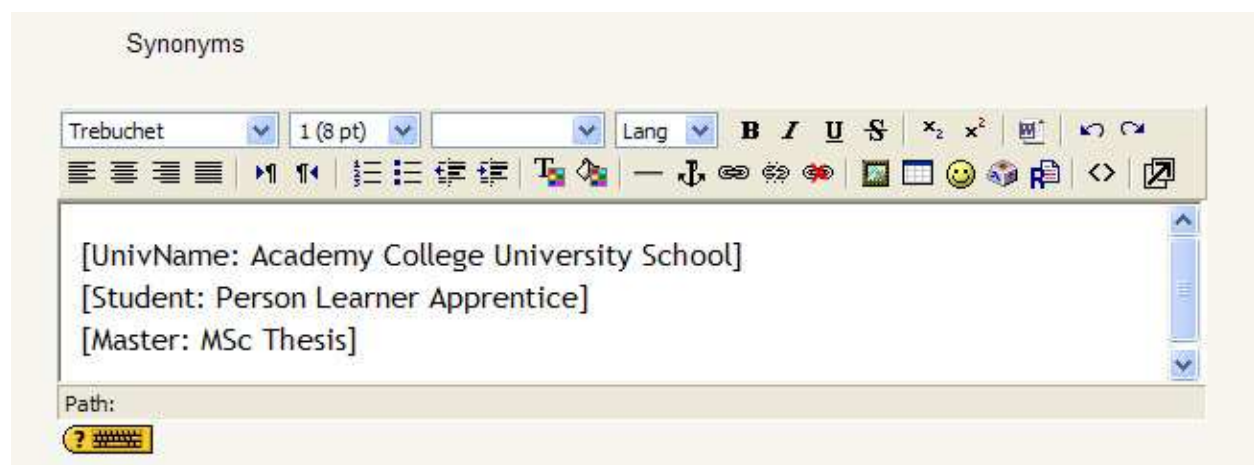
Fig. 11. The `synonyms` html editor element visible in the teacher's window.

To define alternative words for *UnivName*, *Student* and *Master* tags, teacher should prepare following script:

```
[UnivName: academy college university school]
[student: person learner apprentice]
[Master: MSc thesis]
```

Such script has to be placed within the *synonyms* element when task is being defined or edited. Subsequently, it is analyzed by the *validation* function, in which the operation of additional HTML symbols removal comes first. Next, each row of the script is being prepared to be loaded into the database. Two versions of database structures for storing synonyms were analyzed. On one hand, they can be inserted into one column and treated as one text. On the other, they can be spread into two tables - one being a kind of synonyms dictionary with one record per synonym and second one associating tags with their synonyms. This solution allows for writing a synonym once and then map it to many tags, but requires more tables' joins than in the first case. However, both of them were implemented and comparing their effectiveness is planned in the further research.

The simplified algorithm of the procedure for validating participant's solution correctness, which takes mechanism of synonyms management into account, is shown in the listing 1.
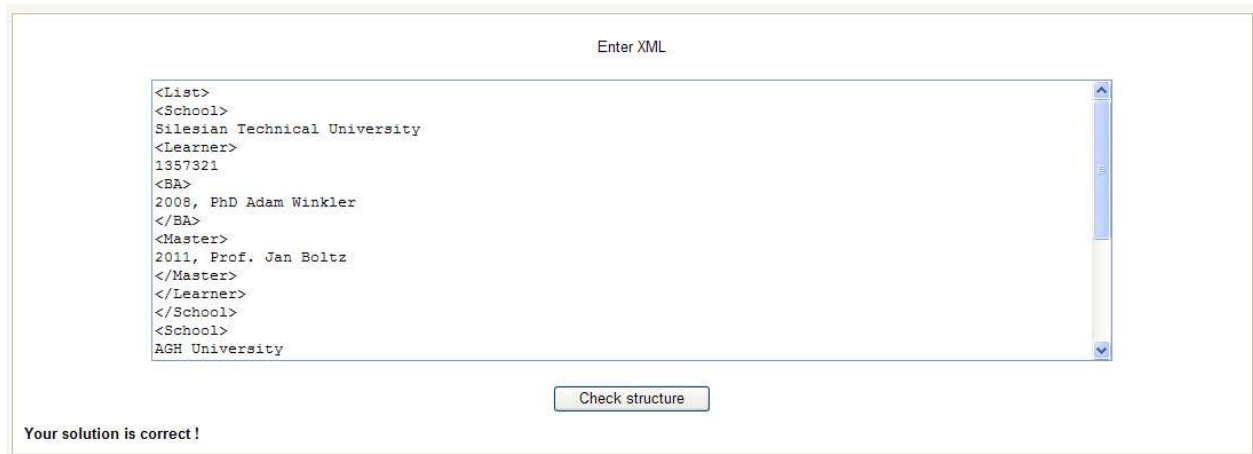

Fig. 12. View of a participant's window with a solution using synonyms.

```
get participant's solution;
while (next_line_read){
    check participant_tag in the tags_template table;
    if (not found){
        check participant_tag in the Synonyms table(s);
        if (not found){
            show error message;
            set error_var;
        }else
            get tag for a participant_tag;
    }
    if (not error_var){
        check a level of participant_tag;
        if (level_error)
                show error message;
    }
}
```

Listing 1. The simplified algorithm of validating participant's solution correctness.

Thanks to the developed synonyms management functionality, mechanisms for validation of participants task solutions become more flexible. In case when function, which implements this functionality, discovers differences in template and student tags, it can still consider a given solution as a correct one (Fig. 12) if it finds a tag, provided by a student, in the synonyms table/tables.

Moreover, in case of a need, a once-defined set of synonyms can be extended by usage of the mechanism described in subsequent parts of the chapter.

5. Creation of an educational path

Process of acquiring necessary skills can develop at a different speed for different course participants. Sometimes presented knowledge requires only few tasks to be done, in other cases it should be practiced many times. For this reason, the simple mechanism, determining the number and difficulty of tasks to be solved by a given participant, depending on his/her

current progress, was proposed in the developed activity. In the present stage of the research, it is based on a number of attempts allowed to obtain a proper solution of a task. This number should be specified by a teacher during defining the task (`attempts_number` in **mod_form.php** file - as it was presented in the fig. 10), but if it is omitted, the default value is taken into account.

This value is calculated for a participant accordingly with its activity (column *Attempt* in the *Grade* table) and is used by the logic of the **view.php** file to check how he/she dealt with a given task. Among cases being considered by this logic, the following can be enumerated.

1. Course participant has not attempted to solve a task yet - as a result, task window is opened for a task defined by a teacher.

2. Course participant has already solved a task - an appropriate message and three buttons are displayed; one for task redoing, second for solving a new task, with the same difficulty level and third for cancelling the action (Fig. 13).
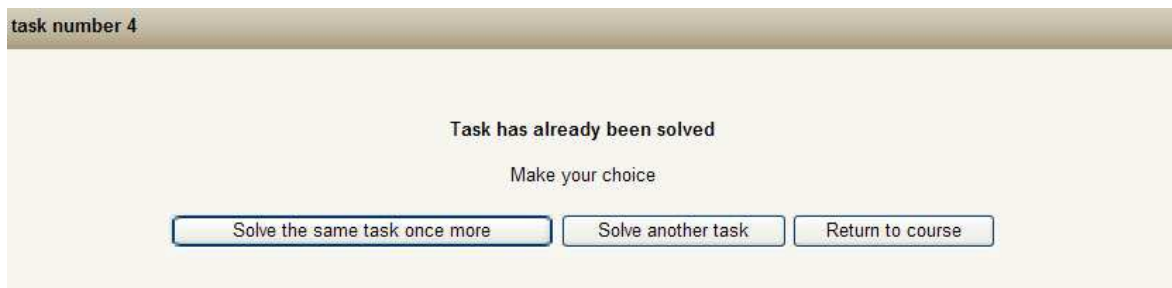
Fig. 13. View of a participant's window with a given task already solved.

3. Course participant undertakes the next attempt to solve a task, which previously was not solved correctly, but an attempt number is less than the highest acceptable (e.g. less than defined by a teacher) - in such situation, acting of the **view.php** file logic is the same as in the first case.

4. Course participant undertakes the next try to solve a task but an attempt number is greater than acceptable - discovering such situation results in displaying a message, which includes task number that should be completed before a next attempt to the current one occurs (Fig. 14).
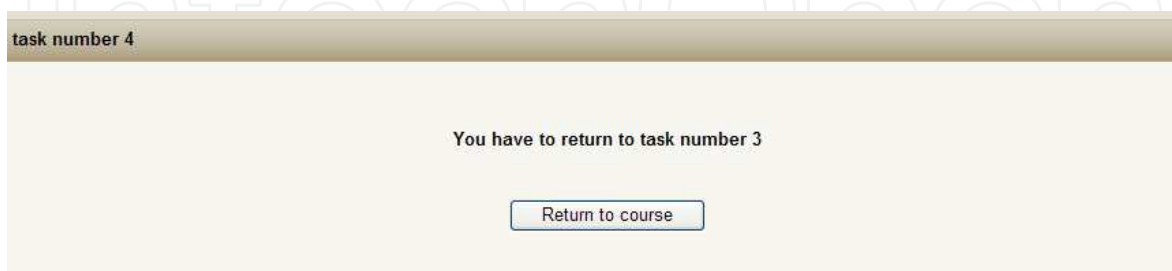
Fig. 14. View of a participant's window with a given task not solved properly.

5. Course participant gets back to solving a task, because he/she didn't manage to solve one of the subsequent tasks. In such case, the subject of the task should be changed, but its difficulty should stay on the same level. Activity of this kind entails creating a new template solution and preparing a new set of synonyms. That process can be automated

by usage of text files with standardized names. Their responsibility is to define new contents enabling the configuration of the task in accordance with the attempt number. The basic part of the file name (so called *prefix*), together with its path, should be provided by a teacher, in task definition window through `file_name` element (Fig. 15).

```
$mform->addElement('text','file_name',get_string('file name','xml'));
```
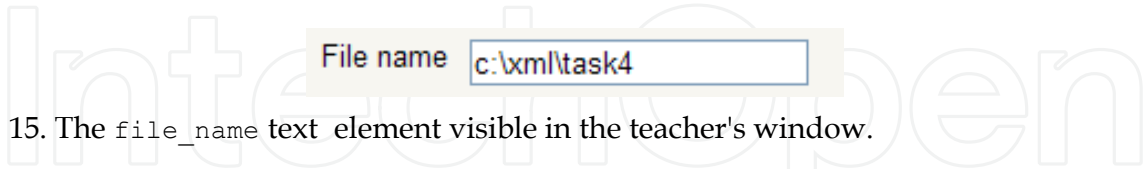
File name c:\xml\task4

Fig. 15. The `file_name` text element visible in the teacher's window.

Its content is then used for constructing full files names. Among them there are:

- *prefix_cont_attemp_number.txt* - containing task subject for a given attempt number,
- *prefix_temp_attemp_number.txt* - containing template of the solution for a given attempt number,
- *prefix_synonym_attemp_number.txt* - containing synonyms for a given attempt number, prepared in accordance with the aforementioned format.

Let's assume that a course participant did not manage to solve the task marked by number 5, because he/she exceeded number of allowed attempts. According to the proposed rules he/she was directed to get back to task number 4. In the first attempt, it is defined to solve the problem presented in Sample XML data structure part, but now the content of the task have to be changed. Let's assume once again that a file *prefix* provided by a teacher has the same value as it is shown in the fig. 15. This means that files used for the second attempt of the task will be searched for in *c:\xml* folder with following conditions:

- *c:\xml\task4_cont_2.txt* - for a new task content,
- *c:\xml\task4_temp_2.txt* - for a new task solution template,
- *c:\xml\task4_synonym_2.txt* - for the new task synonyms.

In case one of these files cannot be found, the first attempt configuration is used.

Simplified algorithm ensuring the described functionality is presented in the listing 2.

```
get passed value for a given task and a given user;
if (passed){
   show message "Task has already been solved" ;
   show appropriate buttons;
}else{
   get attept_number for a given task and a given user;
   If (attempt_number > 1){
       get file_name value for a given task;
       search for appropriate files;
       if (found) {
         define a new task content using prefix_temp_attemp_number.txt;
         prepare a new task solution template using_
         _prefix_temp_temp_number.txt
         insert a new template into a database;
         prepare new task synonyms using prefix_synonym_temp_number.txt
         insert new synonyms into a database;
       }
   }
  open a course participant's window;
}
```

Listing 2. Simplified algorithm of an educational path creation.

The basic structure of teacher's window including all described elements is presented in the fig. 16.
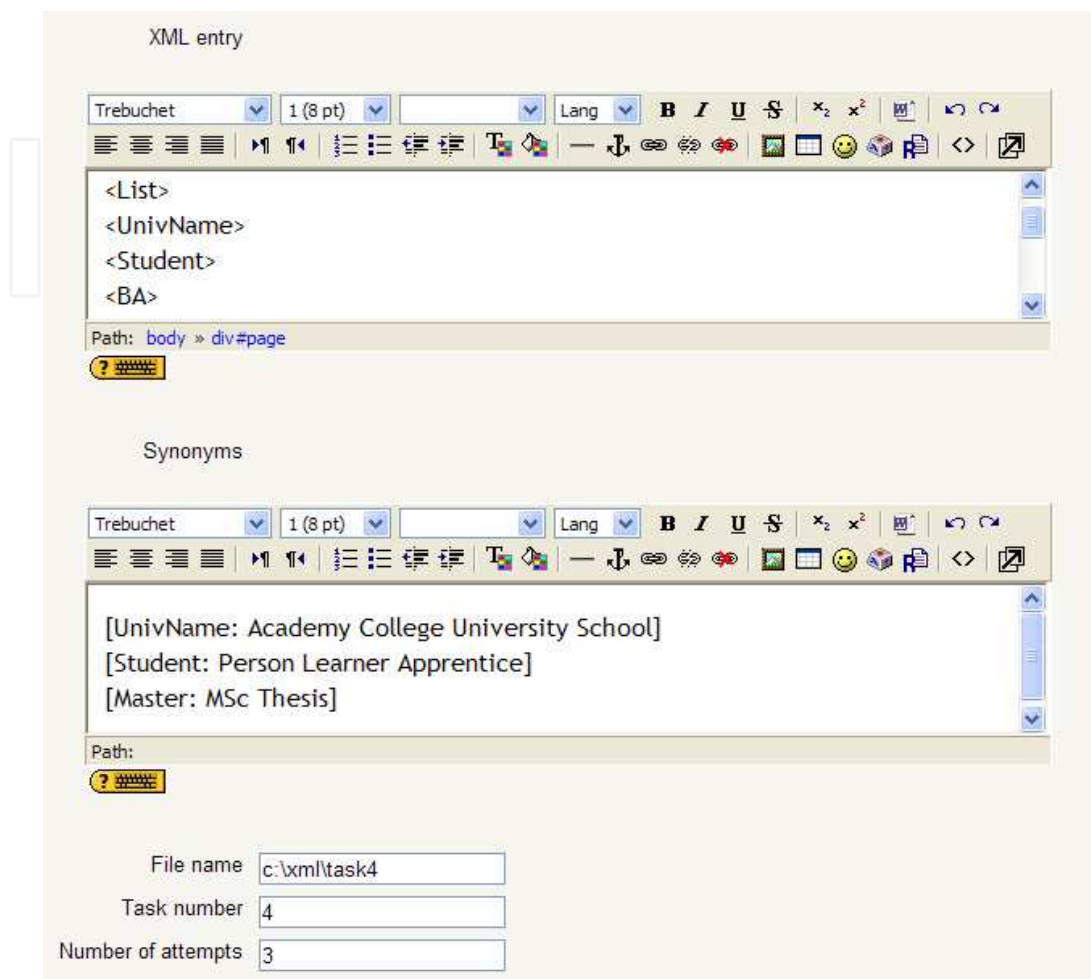

Fig. 16. The teacher's window with all described elements.

6. Advanced mechanisms for the XML activity

The basic functionality of XML activity was tested in terms of its usability. The results of tests indicated the possibilities of further e-learning platform extension. One of them regarded the possibility of the earlier defined set of synonyms extension. Besides, they concerned more advanced XML language issues, directed to the group of participants being at a higher education level.
Therefore, the suitable mechanisms for these inconveniences were proposed.

6.1 The synonym block

Another proposal to solve the difficulty of XML tags discrepancy is to use a specially implemented *Synonym* block. This mechanism provides the e-learning platform with the functionality for defining words and their synonyms, as it is shown in fig. 17.
There were several reasons for the choice of this type of the MOODLE environment component. The main one was the possibility of adding this kind of object to any module and any course, and, hence, achieving an opportunity to its later usage in other than the

XML modules. The SQL modules that were developed in the previous research (Harężlak&Werner, 2010) can be a good example of such situation.

Fig. 17. The Synonym block.

In that case, the synonym block might be useful in situations, when the schema of a database queried by learners by the usage of SQL module, would change. Then, the typical action performed by a course trainer, would be to rewrite all query tasks (i.e. SQL activities) added to the topic previously and to change the names of the altered columns to a new ones. Whereas, giving the teachers the ability to use the synonym block element, would eliminate this disadvantage.

In order to define a new block, the */blocks* folder of the MOODLE home directory was changed. As it was mentioned before, the MOODLE software has the modular structure and all its units have similar construction. The same applies to the block element. In practice, in order to define a new synonym block, a new subfolder with the name corresponding to the name of being defined block, with a known set of PHP and HTML files is created. The chosen, mandatory, elements of */synonym* folder, are listed in table 2.

Chosen elements of /synonym folder	Content description
/db	Defines the structure of back-end MOODLE database table, that is created especially for defined block. It is used during block installation
/images	Includes two icons shown in fig. 17 that are next to the links forwarding the user to word or synonyms defining pages
/lang	Includes subfolders with translation files that consists of string mappings, realizing different language versions of block fields' headers
block_synonym.php	Provides block class definition with functions displaying – inter alia - the title and version of synonym block
synonym.php	This file name must be the same, as the name of defined block

Table 2. The elements of /synonym folder.

Besides, there are a few HTML files that provide block class, in */synonym* subfolder with specific - instance or global level - configuration.

The most important file is the **block_synonym.php**, where - inter alia - two Web links, referencing to separate PHP files, providing the desired functionality, are specified:

```
$this->content->items[] = '<a href="word.php">Store new word</a>';
$this->content->items[] = '<a href="synonym.php">Store synonym</a>';
```

Thus, the major modifications of the new block functionality were made in above mentioned files. It is worth emphasizing that these files are located both - in MOODLE home directory and its */course* subfolder. Such duplication was necessary to make files accessible for MOODLE users both - at the site, and the course, levels (Fig. 18).
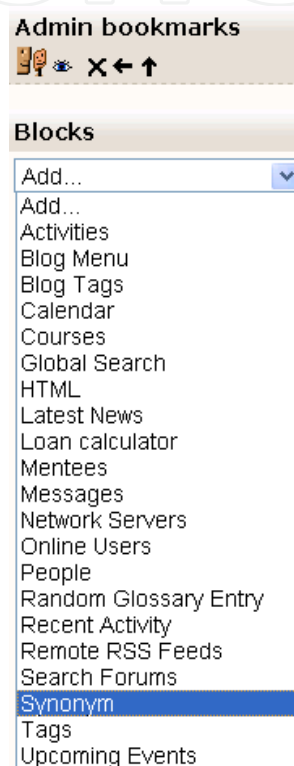
Fig. 18. Adding the Synonym block at the site level.

New added data - i.e. word itself, word id, annotation and the number of course, the word is defined for is inserted into described earlier synonyms table. The last attribute was intended to enable the word definition only to the selected course that – consequently – reduces the number of searched table rows. New rows are inserted to these table by the use of the especially prepared files - **word.php** and **synonym.php** ones.

The existence of two separate PHP files causes that in an independent manner - i.e. not necessarily at the same time - words and their synonyms can be defined.

The result of the **word.php** file acting is presented in fig. 19.

6.2 DTD documents and XML schema

As it was shown, the validity of XML document can be checked by a lexical analyzer. But in terms of the W3C organization (XML Essentials) valid XML document is a well formed XML document, which also conforms to the rules of a Document Type Definition (DTD) or by its successor – XML Schema.

Fig. 19. Defining a new word for synonyms window.

The DTD may be declared inside the XML file (then it should be wrapped in a *DOCTYPE* definition) or in an external file (then, XML file should be assigned to the DTD definition by `<!DOCTYPE root-element SYSTEM "DTDfilename">` element). There is the possibility to extend MOODLE platform with required DTD or XML Schema validation functionality (Webmaster Tutorial – PHP Tutorials), however such validators are also available on-line. As an example, http://www.w3schools.com/xml/xml_validator.asp web page is presented in fig. 20.

Fig. 20. The sample of two-step XML Schema validation window.

In addition to XML instance against DTD/XML Schema validation, another important problem was solved in the described XML module.

In order to provide mechanisms for comprehensive study of the rules regarding constructing of DTD and XML Schema documents, the suitable algorithms were implemented.

For example, one of the module tasks is to study the sample XML instance, written in a task body and correct given DTD document to fit the specified XML. The mechanism which had

to be implemented in the MOODLE platform to check the coincidence of learners' responses with the "model" answer defined by a teacher, was coded in PHP by the usage of auxiliary files.

To grade the correctness of trainee solution (Fig. 21), the sample (i.e. accurate), well-formed, DTD document defined by a teacher is uploaded into a supplementary text file, while the sent answer - into the second temporary text file.

Fig. 21. The DTD correctness task window.

Both files are stored in the MOODLE backend server. Because of the fact, a lot of course participants can execute the task at the same time, the temporary file name generated for a particular MOODLE user solution is dynamically composed in a following manner:

XML_*UserId_ModuleId_TaskNo*.txt.

UserId, *ModuleId* and *TaskNo* represent course participant, module and task unique identifiers respectively and are accessible through PHP variables `$XML->dblogin`, `$course->id`, `$XML->id`, where `$XML = get_record('XML', 'id', $cm->instance)`.
Additionally, to avoid the situation, the numerous spaces will influence the solutions discrepancy, the undue spaces and whitespaces are eliminated from sent answer by the use of regular expressions in form of `preg_replace("@\s+@"," ",$file)`. Generally, the whole algorithm is as follows:

```
Set variables, pointing to both files;
Open files for reading;
Get the contents of both files;
Remove undue spaces and whitespaces from the file contents;
For (each DTD entry) do {
  Compare an entry with its exemplar;
  If (the entries are the same) {
    Return the message of successful task solving;
  }
  Else {
    Return the message of incorrect task solving; }
}
Close the files;
```

Although the XML Schema defining rules differ from the DTD ones, the problems that had to be solved by providing the MOODLE environment with adequate mechanisms were similar. But it must be emphasized, that XML module tasks referring to XML Schemas play another, crucial, role in the topic of storing XML in relational database. Thus, numerous of the new XML functionalities were formulated with regard to a database used in this scope.

7. XML usage in relational databases

Nowadays, a lot of relational databases – e.g. Oracle and MS SQL Server – provide XML Schema support for the numerous purposes. Validation of the XML documents against registered XML Schema definitions is one of them. As a part of the XML Schema registration process, a database automatically creates the storage for a particular set of XML documents, based on the information provided by the schema. If the document fits to the schema, it is shredded into relational data conforming to the underlying relational table.
In described XML activity, the schema registration functionality was introduced. To enable participants the registration of their own XML schemas, suitable PHP script was added to a **view.php** file. In this type of tasks, the *Execute* button is used to perform the requested action, as it is shown in fig. 22.
In fact, the registration process involves an appropriate built-in procedure. For example in Oracle database repository it was *DBMS_XMLSCHEMA.RegisterSchema* procedure. Consequently, in the **view.php** file, the learner schema stored in a string variable, is only
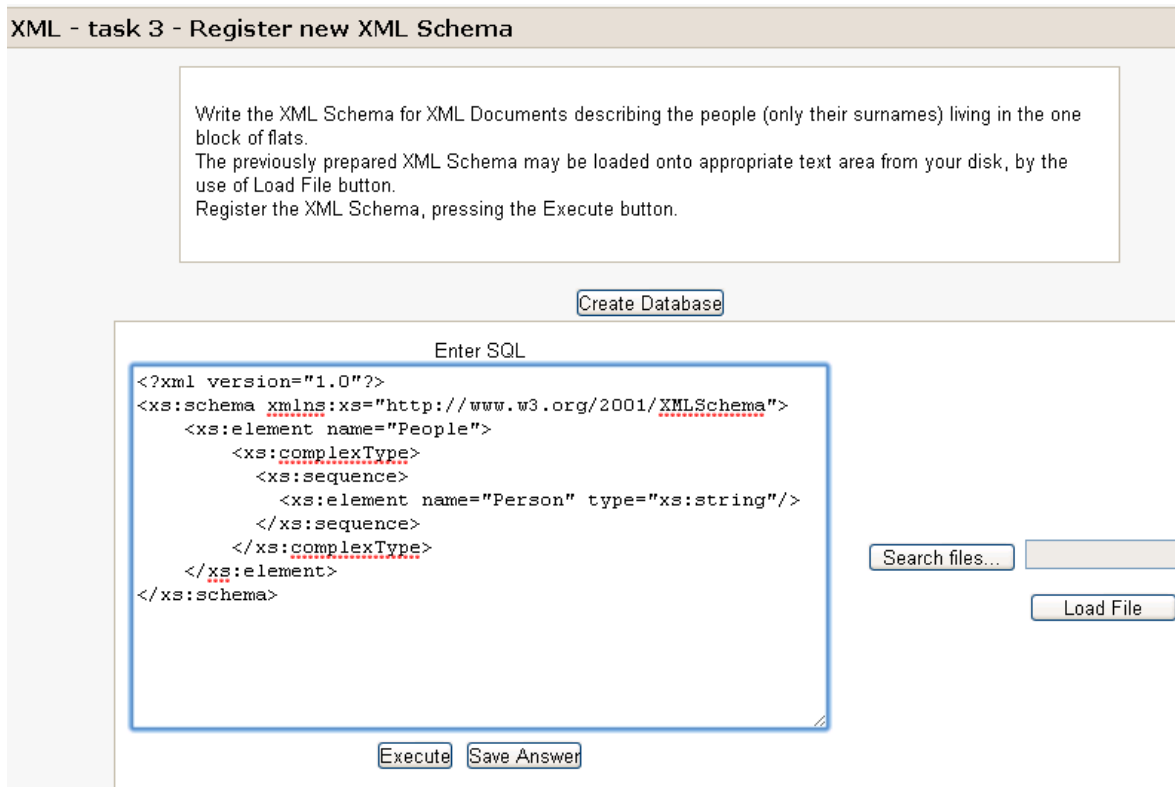
Fig. 22. The sample XML Schema registration.

one of the required procedure arguments. Therefore, after pressing the *Execute* button in the XML module window, the following code is performed (listing 3):

```
$conn = oci_connect($dbuser, $dbpassword, $dbname);
If ($conn) {
  $sql_statement = OCIParse(
    $conn,
    "begin DBMS_XMLSCHEMA.RegisterSchema (
      $SchemaName, $userschema, TRUE,TRUE,FALSE,TRUE, $SchemaOwner);
    end;");
  $command=oci_execute($sql_statement);
  If ($command) {
      echo "Schema is registered successfully";
  }
}
```

Listing 3. PHP code of XML Schema registration.

The schema registration is only the first step to enable storing the XML data in a relational database. Next, it can be, for example, schema-based table creation, as shown in the following example:

```
CREATE TABLE XmlTable OF XMLType
  XMLSCHEMA "http://www.XMLModule.com/sample.xsd"
  ELEMENT "List";
```

or posting a schema-based document to be inserted into a schema-based XMLType view. In this case, the database system checks whether the XML document being inserted into the

table or view, conforms to the XML schema on which the view is defined. To illustrate this process, two tables in Oracle database - *Student* and *University*, with fields corresponding to tags of sample XML (Fig. 2), were created. Additionally, a new *univNo* column was added to each table, in order to enable table joining. The appropriate view definition is shown below (Listing 4).

```
CREATE OR REPLACE VIEW univ_v OF XMLType WITH OBJECT ID
    (EXTRACT(OBJECT_VALUE, '/UNIVERSITY/@univNo').getNumberVal())
AS SELECT XMLELEMENT
    ("UNIVERSITY",
        XMLFOREST(univNo, univName),
        (SELECT XMLELEMENT("STUDENT",
            XMLAGG(
                XMLELEMENT("STUDENT",
                    XMLFOREST(studNO as "Student", BA , Master)
                )
            )
        )
        FROM Student s, University u WHERE s.univNo = u.univNo
        )
    )
FROM University u;
```

Listing 4. XMLType view definition.

There are two ways to keep data in a XMLType storage structure, in Oracle database system:
- Store XML in CLOB XMLType.
- Store XML as structured data, by the use of object-relational storage. In this situation, the shredded XML document is inserted into the underlying relational table, as a new row.

In the **view.php** file only the second possibility was implemented (Building Database-Driven PHP Applications), as it is shown in listing 5.

```
$sql = $_POST["query"];
$tmp = str_replace(";", " ", $sql);
$sql = $tmp;
$sql_statement = OCIParse($connection,$sql);
oci_execute($sql_statement);
If ($r) {
  oci_fetch($sql_statement);
  $strXMLData = oci_result($sql_statement, 'RESULT');
  $doc = new DOMDocument("1.0", "UTF-8");
  $doc->loadXML($strXMLData); }
```

Listing 5. XML data result.

It is worth emphasizing that XML data retrieved from the database with applied DOM fidelity consists of the same information as it was inserted into the database, with the exception of insignificant whitespaces.

In order to print out the result in a proper way (i.e. XML-formatted), the *htmlspecialchars($strXMLData)* function was called. Otherwise, only "pure" data – i.e. entries without tags, will be printed. The XML-formatted result of a sample query is shown in fig. 23, while "ordinary" one – in fig. 24.
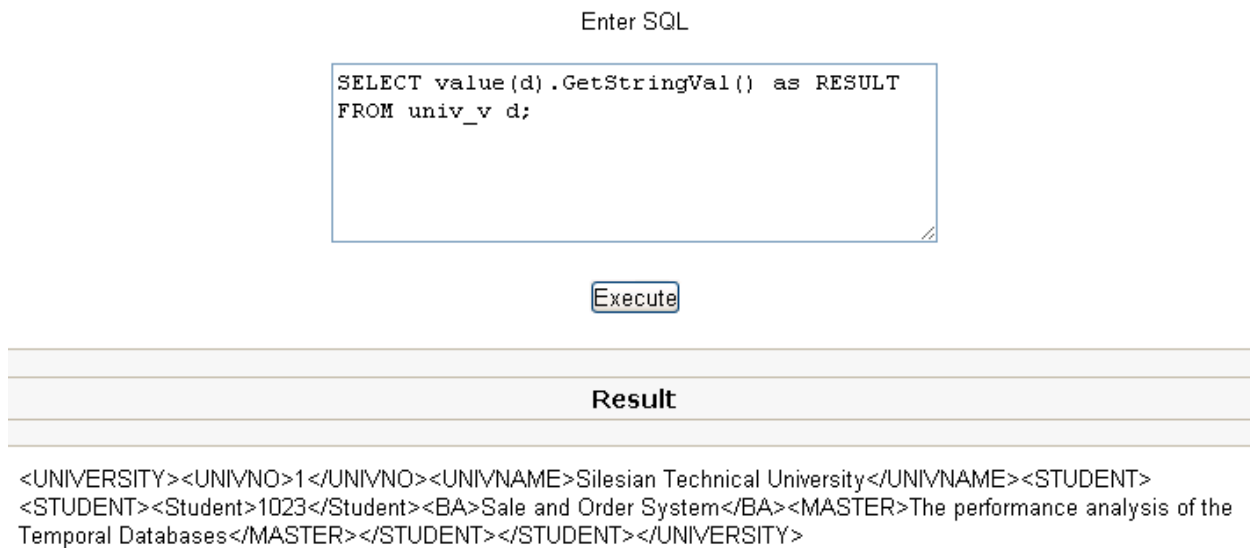
Enter SQL

```
SELECT value(d).GetStringVal() as RESULT
FROM univ_v d;
```

Execute

Result

<UNIVERSITY><UNIVNO>1</UNIVNO><UNIVNAME>Silesian Technical University</UNIVNAME><STUDENT>
<STUDENT><Student>1023</Student><BA>Sale and Order System</BA><MASTER>The performance analysis of the
Temporal Databases</MASTER></STUDENT></STUDENT></UNIVERSITY>

Fig. 23. The XML-formatted result of performing the sample XMLType-based query.

Result

1Silesian Technical University1023Sale and Order SystemThe performance analysis of the Temporal Databases
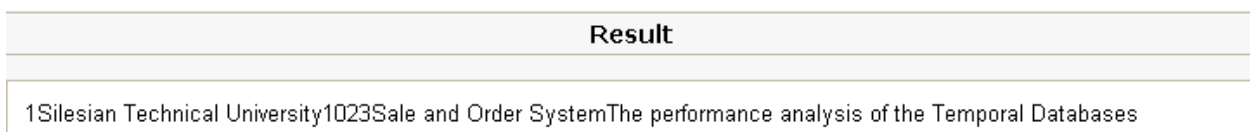
Fig. 24. The result of performing the sample XMLType-based query.

Many other mechanisms, required for manipulating and querying the XMLType tables or views, based on registered XML schemas, were provided to the MOODLE e-learning platform. There are a lot of PHP scripts examples in (PHP Oracle Web Development) that are in fact ready-made solutions. Thus, the numerous functionalities were introduced into the XML module.

Besides, the possibility of formulating a simple XPath queries is provided in a described module. As it is presented in fig. 25, the queried XML document is assumed to be stored in a MOODLE platform server.

In order to provide such functionality, a special PHP function that converts an XML file into an object should be utilized. By the use of:

```
$doc =simplexml_load_file('XML_doc.xml');
```

assignment, all document elements are available. For example, in order to print BA-tagged data, `$doc->student->BA` argument should be used in the *printf* function. To facilitate the algorithm implementation, the document tree hierarchy is stored in a specially created table. Thus, the XPath query realization is reduced to the following steps:

- Check the name of last-written tag.
- Search the auxiliary table to get the level of the tag.
- Convert the well-formed XML document in the given file to an object.
- Loop through the elements at a specified level.
- Print required data in a user's activity window.

Fig. 25. The result of the sample XPath query.

8. Conclusion

In the chapter the possibility of using e-learning platform as an environment for education of XML language issues has been analyzed. Among them, basic XML structure elements and XML instance analysis were considered. For this purpose appropriate XML module and Synonym block, were created.

The architecture of the module is similar to other built-in modules, but its functionality entailed introducing the specific solutions, suitable for tasks that XML module should implement. Additionally the simple mechanism, determining the number and difficulty of tasks to be solved by given learners, depending on their current progress, was proposed in the developed activity. It is based on numbers of their attempts undertaken to obtain proper task solution. Besides, more advanced XML language issues, directed to the group of participants being at a higher education level, were taken into consideration as well. Thus, retrieving relational data as XML and querying XML data with usage of advanced methods were implemented.

All of the extensions were made using PHP scripts and were tested in the chosen database servers: MySQL, MS SQL Server and Oracle DBMS.

Results of all tests were satisfactory, confirming preliminary assumptions for the possibility of using e-learning MOODLE platform as the environment for interactive teaching of XML language issues. However, there are some limitations in the proposed solution. One of them is the necessity to define the XML elements' attributes as the independent tags in XML instance and cannot be placed within other tags. Besides, there is no possibility to attach externally parsed entities to an XML document. Developed functionality does not cover the subject of XML Queries, either.
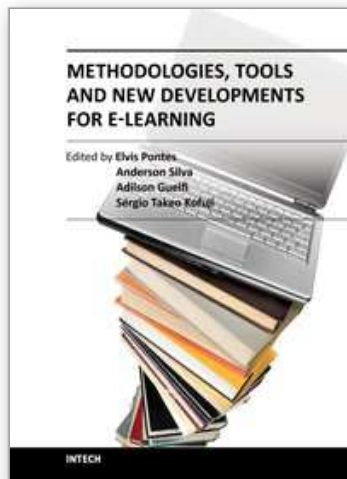
Elimination of these limitations as well as evaluation of prepared solutions for wide group of beneficiaries are the first steps of the future work. Next, there is an addition planned to the presented work, extending grader report module with mechanisms assessing solutions of XML tasks. Furthermore, it is worth mentioning that the proposed functionality regards XML language issues but some of its mechanisms are generally enough to be exploited in other areas of education. The lexical analysis and text management (for instance in subject of programming languages) are good examples of such fields.

9. Acknowledgment

10. References

Braccini, A.M., Silvestri, C., Za, S., D'Atri, A. (2009). Users' perception of open source e-learning platform. Quality: the case of MOODLE. Available from http://eprints.luiss.it/934/2/Users_perception_of_open_source_e-learning_platform_quality_the_case_of_moodle.pdf

Bray, T., Paoli J., Sperberg-McQueen, C. M. (1998), eds, Extensible Markup Language (XML) 1.0, W3C Recommendation, Available from http://www.w3.org/TR/1998/REC-xml-19980210

Daku, B. (2009) Individualized Laboratory Using Moodle. FIE'09 Proceedings of the 39th IEEE international conference on Frontiers in education conference, ISBN: 978-1-4244-4715-2, NJ USA, 2009.

Dobrzański, L.A., Brom, F., Brytan, Z. (2007). Use of e-learning in teaching fundamentals of materials science. Journal of Achievements in Materials and Manufacturing Engineering, Vol. 24, ISSUE 2, October, 2007.

Harężlak, K., Werner, A. (2009). E-learning Database Course with Usage of Interactive Database Querying. Springer Verlag, *Internet - Technical Development and Applications Series: Advances in Intelligent and Soft Computing*, Vol. 64. Tkacz, E., Kapczynski, A. (Eds.), pp. 81-89, Softcover ISBN: 978-3-642-05018-3, Berlin 2009

Harężlak, K., Werner, A. (2010). Extension of the MOODLE e-learning platform with database management mechanisms. *Proceedings of 3rd Human System Interaction (HSI),* pp. 491-495, ISBN 978-1-4244-7560-5, Digital Object Identifier 10.1109/HSI.2010.5514525, Rzeszów, May, 2010 http://www.w3.org/standards/xml/core

Martin, R. C. (2002).Agile Software Development, Principles, Patterns, and Practices. Prentice Hall Publ. ISBN-10: 0135974445, ISBN-13: 978-0135974445, 2002.

MOODLE Home Page - About MOODLE, Available from http://moodle.org

MOODLE Home Page - Statistics, Available from http://moodle.org/stats/

Nedeva, V. (2005). The possibilities of e-learning, Based on MOODLE software platform. Trakia Journal of Sciences, Vol. 3, No. 7, pp. 12-19. ISSN 1312-1723, 2005. Available from http://tru.uni-sz.bg/tsj/vol3No7_1_files/V.Nedeva.pdf

O'REILLY xml.com, Available from http://www.xml.com/pub/rg/XML_Editors

Rice, F. (2006) Word 2007 XML Format, Microsoft Corporation, Available from http://msdn.microsoft.com/en-us/library/bb266220(v=office.12).aspx

Wasiliev, Y. Building Database-Driven PHP Applications on Oracle XML DB, Available from http://www.oracle.com/technetwork/articles/vasiliev-xmldb-php-094969.html

Wasiliev, Y. PHP Oracle Web Development – Data processing, Security, Caching, XML, Web Services, and AJAX. (2007). PACKT Publishing, Retrieved from http://packtlib.packtpub.com/library/9781847193636/ch08lvl1sec02

Webmaster Tutorial – PHP Tutorial, Available from http://www.webmaster-tutorial.com/tutorial/PHP/XML/1

XML Essentials, Available from http://www.w3.org/standards/xml/core

Methodologies, Tools and New Developments for E-Learning
Edited by Dr. Elvis Pontes

With the resources provided by communication technologies, E-learning has been employed in multiple universities, as well as in wide range of training centers and schools. This book presents a structured collection of chapters, dealing with the subject and stressing the importance of E-learning. It shows the evolution of E-learning, with discussion about tools, methodologies, improvements and new possibilities for long-distance learning. The book is divided into three sections and their respective chapters refer to three macro areas. The first section of the book covers methodologies and tools applied for E-learning, considering collaborative methodologies and specific environments. The second section is about E-learning assessment, highlighting studies about E-learning features and evaluations for different methodologies. The last section deals with the new developments in E-learning, emphasizing subjects like knowledge building in virtual environments, new proposals for architectures in tutoring systems, and case studies.

INTECH
open science | open minds