Office Skill Assessment Project

Design Document

Version 3.1

10 February 2009

Preface

The office skill assessment project started in April 2007 when two bachelor students – Bert Wolters and Rick van Nierop - of the Delft University of Technology (DUT) worked in Sri Lanka for an internship at the e-learning center of the University Of Colombo School Of Computing (UCSC.) This project – the Office Skill Assessment Project (OSAP) – is part of the eBIT project, which is a collaboration project between the DUT, the Stockholm University and the UCSC to develop a Bachelor of Information program based on e-learning. After this three months internship the development of the Office Skill Assessment tool continued at the DUT with a small group of IT students: Bert Wolters, Peter de Klerk, Kristian Slabbekoorn, Egbert Bouman and Martijn Reijerse. They finished the tool in December 2008 and delivered to the UCSC for use and maintenance in the future. This document will present the technical design of the tool. Along with the usual sequence, activity and class diagrams also a lot of documentation has been added about the principles and ideas behind the design to give the reader fully understanding about the design.

This document does not contain the documentation about the Moodle plugin and user interface. Please read the document made by Egbert Bouman for this part of the program.

Contents

1. Abstract

At the University of Colombo School of Computing (UCSC) a new e-learning based Bachelor of Information program has been set up. One of the given courses is focused on the Office skills of the student. For this course a new tool should be developed to assess practical tests automatically, so the teacher does not have to assess tests manually. This report is focused on the assessment of skills in Microsoft Word, Excel and Powerpoint. The feasibility of this is investigated and an approach has been found. A tool has been developed, which can be used to assess the word-, spreadsheet- and presentation- processing skills described by the International Computer Driver License (ICDL) standard.

First a procedure for taking a test has been made. The test consists of a printed document with some instructions about for example the used letter types. The student should reproduce this document, using his skills. The resulting document will be matched with a model answer, provided by the teacher and assessed using a marking schema. The system has two documents. Now the main problem is to compare these documents.

The documents are stored in Microsoft OOXML format, which can be easily read by the tool. The problem of OOXML is that it is not directly comparable. Different structures can be used to represent the same document and a lot of useless data is present. A new format – the DocumentElement format - has been created which has only one possible representation for the same document. Also all non-ICDL data has been removed. The DocumentElement format is a tree of DocumentElements – like paragraphs, pictures, tables, cells – with properties assigned to these elements – like lettertype, size and color.

To create a DocumentElement from an OOXML file the use of parsing technology has been suggested. Three levels of parsing are needed to create this format. The first level is parsing the XML file into an object-oriented representation. The next level is parsing the object-oriented representation into a raw DocumentElement representation. To do this, the parser will be provided with some expert data. This data contains information about the relevance of the DocumentElements. The elements and properties that are not mentioned by ICDL or not important to the teacher will be removed. The last level of parsing is the DocumentSweeper. Some extra rules are added to the system to remove the last peaces of useless data.

After both documents have been parsed, they can be compared. Using a marking schema provided by the teacher a mark can be given. This part needs some extra investigation, because the used algorithms are exponent ional, so limitations are build in to avoid explosion of execution time. This could cause errors and unjust marks in some cases.

A tool has been developed using this techniques and the results are promising. The differences between the marks given by the teacher and the system are in most of the cases beneath 10%, which is acceptable. The developed tool therefore seems to be useful and quickly applicable at the UCSC.

2. Introduction

In Sri Lanka education is paid by the government. Sri Lanka is a third world country and there is not enough money to give every Sri Lankan the same opportunities of studying at a university. Only the students with the best test result history can study at the university. This is the reason there is a demand for education external to the university to provide education to a larger amount of students. To fulfill this demand the Asia e-Bit project has been launched. This project develops and implements a 3-year external Bachelor of Information Technology programme (eBIT) in Sri Lanka, based on collaborative pedagogical methods and effective use of e-learning practice. The project is carried out by the University of Colombo, School of Computing (UCSC) in collaboration with the Stockholm University (SU), Sweden and the Delft University of Technology (DUT), the Netherlands.

A part of the eBIT program is a course where students practice their skills in Office programs. The aim of the Office Skill Assessment Project's (OSAP) is to develop and implement an automated e-assessment system to test the Office skills of the students. Currently skill assessment is done using multiple choice questions (MCQ) or by validating practices manually. MCQ is too limited for the use of skill assessment. It can only be used to test theoretical knowledge. Also the question rises if an academic grade can be given to a student based on only MCQ. From the pedagogical view, assessing practices is a much better way for examination. But validation of practices is much more complicated. Manual validation requires a lot of manpower, which is not available at the UCSC, where the system will be used. In 2007 there will be 300-400 students, and there numbers are likely to increase in the future. There are only 3 or 4 teachers available for all courses given.

The UCSC tries to use high technology for this project. The use of internet technology and e-assessment improves the accessibility of academic education, especially in the rural areas of Sri Lanka. Also a new centre has been built in Colombo, where students can take tests. The infrastructure for a practical skill e-assessment system is present and available. It is a waste of resources if this system would not be used for the purpose it is built for.

So the need and the resources are available for a new e-assessment system. In this report a system will be presented which can automatically assess office skills of the student.

3. Assignment

3.1 Assignment Description

The assignment can be described as followed:

"Develop a tool to automatically assess Office skills of students as described in the International Computer Driver License (ICDL) standard."

The ICDL is a standard where the basic skills of the use of computers are described. A large section of the ICDL concerns Office skills.

3.2 Technologies

The following technologies has been used to build the tool:

- php (object oriented)
- MySQL
- XML
- HTML
- XSLT
- CSS
- AJAX

4. Analysis

There are two main questions that should be answered in this report:

1. What is the procedure of taking a test? What information should a teacher deliver to both the student and the system? What is the input of the student to the system?
2. How can the system assess a student's test?

4.1 UCSC Testing Procedure

The UCSC has recently opened a test centre. The procedure of testing is as followed[4]. A student makes a request for an e-assessment session. The session will take place on a time chosen by the student, as long as there is capacity available at that time in the test centre. This is in contrast to traditional examinations where all the students take the test at the same time. When the student finishes his test, the system should grade it automatically.

This procedure requires the following:

1. A teacher should provide the system enough information to mark a student's test automatically. A marking schema should be provided. Eventually with a model answer or other data.
2. The system should be powerful enough to mark a student, without intervention of the teacher.

4.2 Skill Assessment Methodologies

Assessing Office skills with Multiple-choice Questions as used in the current system is limited. It assesses only knowledge of Office and not the skills of the student. The easiest and maybe the only way to assess skills is to let the student use the Office programs. The system should assess if the student uses it correctly. Two ways to do this are:

1. Use a monitoring tool, which assesses the actions taken by the student. This can be done using simulation.
2. Assess the result document of the student.

The first way is to create a simulated environment of an Office program. For example: if the assignment is to press a certain button. The monitoring tool registers if the student presses the right button. The advantage of this system is that every action can be assessed. If only the result is assessed it is not certain which actions have taken place to achieve this result. For example a copy-paste action can only be assessed with a monitoring tool.

There are great disadvantages of a monitoring tool in combination with simulation compared to assessing the resulting document:

- The complete Office package should be simulated, which is too complex for this project.
- If a new Office package will be developed, the simulation system will become less useful in too short time.
- There are different ways to perform the same action(s). All the ways should be determined for a fair assessment.

[4] See: Appendix G

Although a monitoring tool is capable of assessing more skills than a result assessing tool, there are too much disadvantages. The use of a monitoring tool is therefore not feasible in the scope of this project. A result assessing methodology is more feasible.

4.3 Chosen testing procedure

After this analysis the following procedure has been chosen:

1. The teacher provides a model solution and a marking schema to the system.
2. The teacher provides the student with the assignment, for example a printed document that has to be reproduced, accompanied with some instructions about the required lettertypes, etc.
3. The student take the test using an Office package
4. The student provides the system with the resulting document
5. The system marks the document

As mentioned in the paragraph “UCSC Testing Procedure” the teacher should provide the system with a marking schema and eventually a model solution or other data. The best solution is to provide the required data with the least effort needed for the teacher. In this report a solution will be presented to this problem where only a marking schema and an example solution document are needed. There is not a lot of effort needed in this case.

Another question is what format of the document should be accepted. In the Microsoft Office 2007 a new format (OOXML) for documents has been developed. This XML format can easily be accessed by other programs, like the assessment tool. The advantage of using this XML format is therefore high. Non-Microsoft packages like Open Office are also able to save documents in this format. The problem is that Open Office does not support Microsoft XML completely. In some cases there are differences. Therefore it is better to use Microsoft Office for this format. There are other formats, like OpenDocument format, which is also XML based. But there is no widely accepted standard, so the choice for OOXML is a bit arbitrary.

4.4 Assessment

The last question is how assessment can be done. With the chosen testing procedure the system has two documents to compare to each other. The problem of assessing can now be narrowed to the problem of comparing two XML documents. The following problems can be expected:

- Two documents that look the same on the outside can have completely different XML representations. For example: in Microsoft OOXML format there are different places to store styling information.
- There is a lot of useless data stored inside the XML documents. There are two classes of useless data:
    - Garbage data. For example: an empty paragraph can be bold, but this styling information will not be used, because there is no text to be made bold.
    - Non ICDL data. A student does not have to know every option of an Office program. Only the skills mentioned in the ICDL standard should be assessed.

5. Design

5.1 Overview

This chapter of the document presents the design of the system. It offers a mechanism to tackle the main problem of the system: comparing two documents. The main focus of this chapter is a textual explanation of the used philosophies. The technical design described in the appendices is based on the ideas introduced in this chapter.

5.2 Skill Assessment mechanisms

5.2.1 Introduction

The core business of the system is comparing two documents. To assess a document produced by a student and give it a mark, the system has to use different mechanisms. In this section these mechanisms are described on a high level, where the focus is not the actual implementation and used algorithms. More important is the philosophies behind the used mechanisms and the problems that arise during the assessment. The assessment core of the system uses four different mechanisms:

1. Parse an Office document to a XML document in object format. (level 1 parsing)
2. Parse a XML document in object format to a new Document object format, which makes the actual assessment easier. (level 2 parsing)
3. Sweep the Document object, by removing irrelevant data and combining data where possible.
4. Create a marking schema and compare the teacher's and student's documents, using this schema.

These mechanisms and the used algorithms are described in the next paragraphs.

5.2.2 Level 1 parsing

The system makes use of Open Office XML File Format (OOXML). This file format is common used in the Microsoft Office 2007 package (for example in Word the files saved in this format has the docx extension.) This file format has some important characteristics:

1. The document is saved in different XML formatted files.
2. Every file has an own domain. Some files describe the styling, other the contents and another kind of files the relationships between the files.
3. The tree of files has been zipped into a docx file (for the Word-example)

So, the first level of parsing has different jobs to do. For the third problem, a standard unzip-class in php has been used. After unzipping a directory tree has been created. If you store a simple Word file the following tree will be obtained:

- Worddocument.docx
  - _rels
    - .rels
  - docProps
    - app.xml
    - core.xml
  - word
    - _rels
      - document.xml.rels
    - theme
      - theme1.xml
    - document.xml
    - fontTable.xml
    - numbering.xml
    - settings.xml
    - styles.xml
    - webSettings.xml

Figure 1: a docx file unzipped

The system has been designed in an early stage for the older Microsoft 2003 XML format. The main difference between the old and the new format is that in the older version everything was stored in 1 xml file and not in several. To use the 2003 engine the system should first make again 1 file of the tree. Important is to see the relationship files (.rels). In this files all the information is provided to concat the files in the right way.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId3"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings"
      Target="settings.xml" />
    <Relationship Id="rId2"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles"
      Target="styles.xml" />
    <Relationship Id="rId1"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/numbering"
      Target="numbering.xml" />
    <Relationship Id="rId6"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme"
      Target="theme/theme1.xml" />
    <Relationship Id="rId5"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable"
      Target="fontTable.xml" />
    <Relationship Id="rId4"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings"
      Target="webSettings.xml" />
  </Relationships>
```

Figure 2: the document.xml.rels relationships file

After the concatenation we have one xml-file. The final step of the first level of parsing implies nothing else then just copy the data from a XML-tag to an object in the object oriented environment. Consider the next example:

```
<w:body>
  <w:p>
    <w:pPr>
      <w:pStyle w:val="Standard"/>
    </w:pPr>
    <w:r>
      <w:t>This is an example</w:t>
    </w:r>
  </w:p>
</w:body>
```

Figure 3: a small part of the document.xml file.

In this example a root object will be created with name "w:body". This object will have one child "w:p", which on his turn will have two children, and so on. The object "w:pStyle" has an attribute, which will be stored in an array. The object "w:t" has some tag data, which will be stored as a string. This is just a straight forward way of parsing. In the system the parser is named "XMLParser" and the object it creates is a "XMLTag". This parser uses the standard php xml parser.

5.2.3 Level 2 parsing

A OOXML document has two important tags. A tag named "styles" containing all the data about formatting the document and a tag named "body" which contains the contents of the document. In the second level of parsing, contents and formatting will no longer be separated in this way. A new object called "DocumentElement" is introduced, which represents by example a paragraph, a table or a text area. This document element has several properties, like the used letter type, whether a text is bold or not or the content of a text area. A document element may also have children, for example a paragraph, which have different text areas.

Another important task of the DocumentParser (this object implements Level 2 parsing.) is to remove data, which is not important for the assessment. Expert data is needed to define whether something is important or not. This expert data is provided to the system by a XML-file. See the following example:

```
<document>
  <body>
    <xmltag>w:body</xmltag>
  </body>
  <text>
    <xmltag>w:r</xmltag>
    <properties>
      <property>PropText</property>
      <property>PropBold</property>
    </properties>
  </text>
</document>
```

Figure 4: a properties XML file

In this document only two document elements are important. These are the body and a text area. Now the system will know that in the original XML document, the document element "body" can be

constructed out of a “w:body” tag and has no properties. The “text” element can be constructed using the “w:r” tag and has two properties: PropText and PropBold. For both of these properties a Property-object will be constructed. For a full syntax of the properties file, see Appendix D.

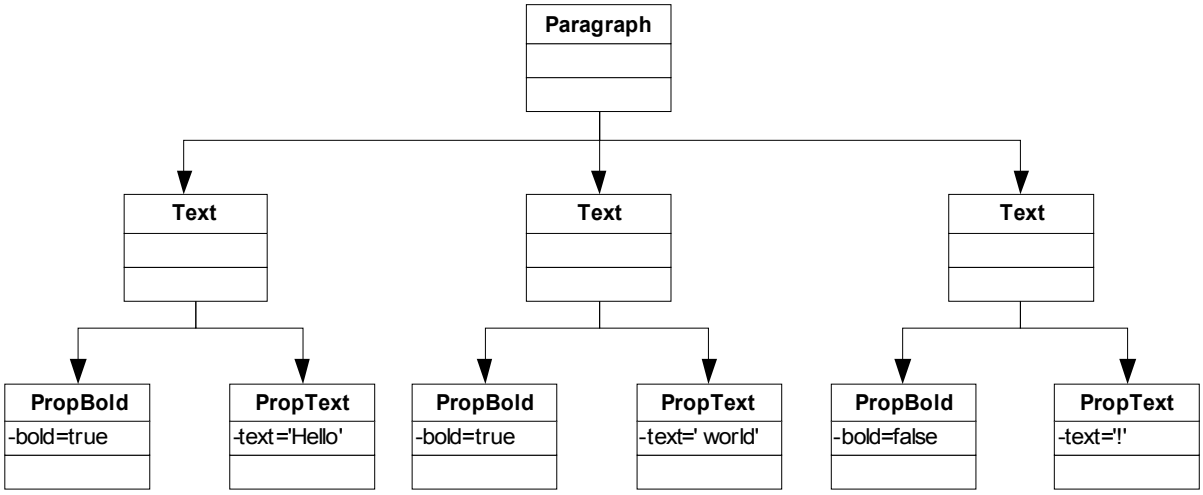


Figure 5: a DocumentElement tree with some properties

Figure 5 shows an example tree that can be constructed from the XML file shown in Figure 4. There is one paragraph, with three text areas. These four objects are all document elements. The paragraph doesn't have any properties. But each of the text objects has two properties. Notice that all the properties available to a document element will be attached to it. So in the case of making a text bold, it is not enough to attach a PropBold object to the document element, but there has to be a boolean value in the PropBold class to show whether the text is bold or not.

Because level 2 parsing is highly dependent on expert data, the system provides an easy and modular way to change the parser to the needs of the expert. If the expert wants to check on a new property (or document element), only the properties XML file has to be changed and a new property objects has to be added to the properties directory. For example: if an expert wants to check on the property 'italic' to see whether a text is italic or not, he has to add the line '<property>PropItalic</property>' to the properties tag of 'text' and add a new class 'PropItalic' to the properties directory, where he has to implement a few abstract methods. The parser will recognize the new property and do his job.

The algorithm used in the implementation is a recursive one. It starts at the root of the XMLtag provided by the level 1 parser and recursively adds children and properties to each document element, using the properties XML file.

5.2.4 Sweeping

A major problem of an OOXML file is that there are many ways to store the same document in this format. Figure 5 shows one problem. The first two text elements have the same properties (except for the text value itself), but are stored in two different elements. They could be easily combined to one element with the text 'Hello world'. The problem is that if the teacher provides a document to the system with one text element and the student provides a document with two text elements, the documents can still be equivalent. So after the level 2 parsing, the document has to be cleaned up by removing irrelevant data elements and combining elements wherever possible, to make assessment possible. For example this sweeping has to deal with the following 4 problems:

1. If all the children of an element have the same property and the element can have this property according to the properties XML file, then this property has to be removed from the children.
2. If the children have different values of one property, then the element may not have this property.
3. If two text elements next to each other have the same properties, except PropText, then these two elements can be combined to one.
4. If a child can have a certain property according to the properties XML file, but no information can be found about this property, copy the parent's property to the child.

The first problem can be the case if a paragraph has only bold text elements. This can be stored in two ways: either the paragraph property can be set to bold, or all the text properties can be set to bold. We choose to say that the paragraph is bold, rather than to say that every text element of the paragraph is bold. To apply this, we remove all PropBold objects of the text elements. Of course, this is only allowed if the expert has turned on this property for a paragraph in the properties XML file.

The second problem occurs for example if in a paragraph some text elements are bold and some not. It would be meaningless to say whether the paragraph is bold or not. So the sweeper will remove this meaningless property of the paragraph if present.

The third problem we discussed before. After sweeping Figure 5 will only consists of two text elements. Now the document is ready to be compared to another document.

The fourth problem occurs often when a property's value cannot be retrieved from the data and a default value cannot be given (for example: text size.) In that case the value of a child has to be ignored (because it is a null-value) and has to be replaced by a parent's value.

The algorithm used for sweeping is also recursive. But it is important not to start with the root, but to start in the bottom leaf. From there the algorithm works his way up to the root. This is important because a document element is depending on the values of his children to be swept correctly. To avoid problems, for sweeping purpose a document element may only use children that are swept before or who has no children on their own.

5.2.5 DocumentParsing against Sweeping

Very important in extending the system with three level parsing is to decide which parser to use for a certain task. Properties and document elements can be parsed in both level 2 (documentparsing) and level 3 (sweeping.) If you want to implement a new property or document element, which parser should be changed? The next characteristics of the parsers can be helpful to decide which parser to use:

DocumentParsing	*Sweeping*
Easy to add new properties/document elements using properties xml framework	Only basic framework provided, designer should provide own code
All information provided in OOXML available	Some information provided in the original file deleted during document parsing
No information available about parsed children/parent tags	Parsed children / parent tags available

Figure 6: Differences between two levels of parsing

The first characteristic shows that in most cases the programmer should use the document parser. It will save a lot of work, because a very helpful and easy to extend framework is provided by the system. The second and third characteristic clearly show when to use sweeping and when to avoid it. Let us consider the first problem described in the previous paragraph:

"If all the children of an element have the same property and the element can have this property according to the properties XML file, then this property has to be removed from the children."

There is no way the document parser could solve this problem. The parser works from top to bottom, so the children of an element are not parsed yet. The parser does not know which properties of the children are important and which are not, if it does not parse them yet. So this is a task of the Sweeper.

On the other hand, if you want to add a property only during sweeping and ignore the document parser, the corresponding OOXML-tag is already destroyed by the document parser, because it will not consider the corresponding tag to be useful. So if you want to use the sweeper, make sure the document parser stores the useful data in some way.

5.2.6 Comparing and marking

Teacher's model

After parsing and sweeping the student's document has to be compared to the right solution of the teacher. Therefore the teacher needs to tell the system how he wants the student's document to be marked. For the marking process a philosophy has been developed. First a model has been made of a teacher assessing manually. It looks as followed:

1. If the teacher finds out that the student made a small mistake, like he used the wrong styling or made a spelling error, the teacher performs the following actions:
    a. The teacher determines the size of the error. Does it matter if the used letter type differs from his? Is it a bad mistake, a small mistake or is the difference not important. After this process the teacher decides how many points will be distracted for this error.
    b. The teacher checks if the student makes the same mistake over and over again. After 3 or 4 times the error maybe punished enough in the eyes of the teacher.
    c. The same error is punished equally through the document. So if a student makes a layout error in the first paragraph, it will be punished with the same penalty points as if the error was made in the third paragraph.
2. If the teacher finds out that the student is missing a complete element, like a picture or a paragraph, the teacher will distract points for the missing element from the total.
3. If the teacher finds out the student has created an extra element, like a picture where it was not needed, the teacher will distract points for the surplus element.
4. The points that can be extracted for errors can exceed many times the maximum grade. If the grade is on a scale of 1 to 10, if the student makes half of the test badly, the grade can be a 1, instead of a 5.5.

At the end the penalty points are counted and distracted from a maximum grade that could be awarded to the complete test.

Implemented model

With the philosophies described here above we can design a new technical model for the DocumentElement/properties structure. Because mistakes in properties are punished equally through the document (1c), it makes sense to define the punishments globally. So in the system for every property a new GlobalProperty will be created. The teacher can set different things, like if the system should check on it and how many points should be distracted if the student makes an error with it (1a). Also a maximum amount of same errors can be defined, to avoid the system punishing the same error over and over again (1b.)

Also another feature has been added. For every property of a specific document element it is also possible to turn assessing on or off. This is important, because sometimes the assignment requires that the student's answer differs from the teacher's one. E.g. if the student has to fill in his name, text assessing should be turned off for that paragraph.

For missing elements (2) another structure is needed. It is no longer suitable to say that every missing paragraph should be assessed the same. Some paragraphs are far more complex than others and the teacher would like to extract more points for missing paragraphs that are complex. So the feature should exist that the teacher can select for every element how many points should be extracted when missing. For surplus document elements (3) it is sufficient to extract a standard amount of points.

Extra properties in a student document element will be ignored. A student will be punished for that at a higher level in the document element hierarchy by missing properties there. This because the student didn't make the mistake, but the missing property is result of the document sweeping.

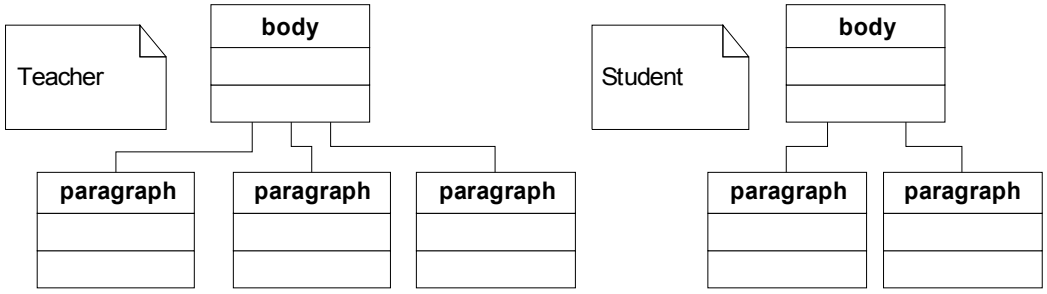


Figure 7: Complex marking example

Figure 7 shows an example of a complex situation. Here the amount of paragraphs differs. To tackle this problem, we need to assign each paragraph of the teacher to a paragraph of the student. This is done by comparing and marking every paragraph of the teacher with every paragraph of the student. Than we optimize this collection so we get the optimal one-to-one mapping of paragraphs. Notice that the original order of the paragraphs in both documents should be conserved. The optimization of this problem was the bottleneck of the previous 2007 version of this project. This because no algorithm could be found that did run in polynomial time order. In the new version an optimization has been found using Linear Programming that runs in quadratic time order.

6. Conclusion

The original assignment was:

"Develop a tool to automatically assess Office skills of students as described in the International Computer Driver License (ICDL) standard."

The following main questions were asked:

- What is the procedure of taking a test? What information should a teacher deliver to both the student and the system? What is the input of the student to the system?
- How can the system assess a student's test?

The following conclusions can be made:

1. The UCSC testing environment requires a testing mechanism where a student can be tested completely automatically. The teacher should supply all the information needed for assessment by the system before a test can be taken by the student.
2. The use of monitoring tools to assess the actions of a student is not feasible in the context of this project. Only assessment of the resulting documents is feasible, although not every action of the student can be recovered and assessed.
3. The Microsoft OOXML format is a satisfying format to extract the necessary information to make skill assessment possible.
4. If the system receives a model document of the solution and a marking schema of the teacher and a resulting document of the student, skill assessment can be done. The effort for the teacher to provide enough information is minimal.
5. The main problem of assessing a student's document with the input described in conclusion 4 is the problem of comparing two documents.
6. The Microsoft OOXML format is not directly comparable, because different structures can be used to represent the same file and a lot of useless data is present. Parsing of the XML document to a directly comparable format is needed for assessment.
7. The OSAP DocumentElement format is a directly comparable format for assessment. This because only one structure can be made to represent one file and only ICDL important data is preserved.
8. A way is presented to parse Microsoft XML to DocumentElement format. Three levels of parsing are needed. This parser is build very flexible. It is easy to add new assessment rules to satisfy the teacher's need. Also extension to other Office programs can be made in a relatively easy way.
9. A marking schema can be provided to the new system. This is based on a real life teacher's assessing model.
10. A tool has been developed witch can assess almost every ICDL Office skill for Word, Excel and Powerpoint. The prototype has been tested and fulfills the requirements.

7. Sources

Lethbridge, T.C., Laganière, R.: Object Oriented Software Engineering second edition, 2005

Moodle coding standard, http://docs.moodle.org/en/Coding

Simpletest, http://www.lastcraft.com/simple_test.php

XML Parser of Adam A. Flynn as seen in http://www.criticaldevelopment.net/xml/. The version integrated into the system is 1.2.0 (2006) and is published under the terms of the GNU General Public License as published by the Free Software Foundation

Design Patterns, www.wikipedia.org/wiki/Design_pattern_(computer_science)

Geers, E.M.A., Asia eBIT – a model for net-based learning to help bridge the knowledge divide, 7 october 2006

Appendix A: Requirements

1. Overview

In this appendix we will summarize the requirements collected by interviewing the parties involved in the OSAP. The requirements are described in the functional requirements (2) and the quality requirements (3). The functional requirements contain the functionality of the system itself. The quality requirements describe how the system will be documented, the maintainability of the system and the date of delivery.

2. Functional Requirements

In this paragraph, the functionality of the system is described. These and only these functions will be implemented in the proposed system. This paragraph is divided into three sections. The first two describe the functionality of the system from the view of the two actors: the teacher and the student. The last section formulates the functionality of the assessment part of the system.

1 *The teacher* creates the tests. For this purpose the following functionality is required:
- 1.1 Sign in into the system using a Moodle teacher's account
- 1.2 Design a test:
  - 1.2.1 Set the name
  - 1.2.2 Add an assignment. An assignment consists of:
    - 1.2.2.1 a model Office Document (Microsoft Office OOXML format) showing the correct solution
    - 1.2.2.2 instructions how the correct solution must be obtained
    - 1.2.2.3 a marking schema
    - 1.2.2.4 a time allocation
  - 1.2.3 Set the time period in which the test can be made
- 1.3 Edit a test:
  - 1.3.1 Edit an assignment (see 1.2.2.1-1.2.2.4)
  - 1.3.2 Edit the time period in which the test can be made
- 1.4 Get a test made by a student
- 1.5 Get the result marks of the test

2 *The student* uses the system to take tests. The following functionality is available:
- 2.1 Sign in into the system using a Moodle student's account
- 2.2 Make a test:
  - 2.2.1 Sign in to a test
  - 2.2.2 Get the remaining time in which the current assignment has to be completed
  - 2.2.3 Get a warning if the remaining time will expire in 5 minutes

2.2.4 Get a warning if the remaining time has expired

2.2.5 Upload a document (Microsoft Office OOXML format) to be assessed

2.2.6 Finish a test

2.2.7 Get the result of the finished test

3 *The system* assesses the documents uploaded by the students. To assess these it should have the following functionality:

3.1 Get the assignment made by the teacher

3.2 Get the document made by the student

3.3 Compare and mark the document using:

3.3.1 the information given by 1.2.2.1-1.2.2.3

3.3.2 expert data of the teacher about what should be considered as right and wrong.

3. Quality Requirements

1 The response time has the following restrictions:

1.1 The actual assessment of a document should not take longer than 5 seconds.

1.2 Uploading documents depends highly on the connection speed and the amount of users trying to upload at the same time. A student may not be punished for having a bad connection.

1.3 All other response times should not take longer than 3 seconds, excluding transmission time.

2 The system should be maintainable and extensible. In the future this system will probably be extended by other project teams. The system should therefore be well documented and clearly structured:

2.1 The followed coding style should be the Moodle Coding Style[5].

2.2 The code should be as object oriented as possible. This include:

2.2.1 Functions and classes are built using the Design by Contract methodology.

2.2.2 Preconditions, postconditions and invariants have to be validated using assertions.

2.3 All functions have to be well tested using the automatic testing environment Simpletest[6].

5 This is described in: http://docs.moodle.org/en/Coding
6 This can be found on: http://www.lastcraft.com/simple_test.php

4. Requirement dependencies

Some requirements depend on other ones. If any of these requirements are implemented in the system, while the requirements they depend on are not, it will be meaningless. For example, if a requirement states that an object can be edited, it would depend on the requirement that an object can be added. All the requirement dependencies are listed below:

	1.1	1.2.1	1.2.2.1	1.2.2.2	1.2.2.3	1.2.2.4	1.2.3	1.3.1	1.3.2	1.4	1.5	2.1	2.2.1	2.2.2	2.2.3	2.2.4	2.2.5	2.2.6	2.2.7	3.1	3.2	MoSCoW
1.1	X																					**W**
1.2.1		X																				**S**
1.2.2.1			X																			**M**
1.2.2.2				X																		**M**
1.2.2.3					X																	**C**
1.2.2.4						X																**S**
1.2.3						D	X															**S**
1.3.1			D	D	D	D		X														**C**
1.3.2							D		X													**C**
1.4										X							D	D				**S**
1.5											X						D	D		D	D	**M**
2.1												X										**W**
2.2.1		D					D						X									**S**
2.2.2						D								X								**C**
2.2.3						D									X							**C**
2.2.4						D										X						**C**
2.2.5																	X					**M**
2.2.6																		X				**M**
2.2.7			D	D	D														X	D	D	**M**
3.1			D	D	D															X		**M**
3.2																	D				X	**M**

Figure 8: Dependencies of requirements. 'D' means: horizontal component is dependent of vertical component.

The last column shows the importancy of the requirement represented by the **M**o**SC**o**W** scale: **M**(ust have), **S**(hould have), **C**(ould have) and **W**(ould have). During the implementation of this system, the focus will be on the requirements on the highest scale of the MoSCoW scale.

Appendix B: Technical System Design

1. Overview

This chapter offers a detailed description of the implementation design. All the used classes are described in UML class diagrams. The most important functions are described in UML sequence diagrams and activity diagrams. Also an explanation of the subsystems is given where necessary. The first paragraphs (2 - 6) give an overview of the core of the system: the DocumentParser and the DocumentMarker. In 2 the top level has been described. The DocumentParser uses three parsers to work properly. These parsers: the XMLParser (3), the TagToElementParser (4) and the DocumentSweeper (5) are documented in this section. In 6 the DocumentMarker with its corresponding Markmaximizer has been described. The last two sections describe the way the User Interface has been build. 7 is focused on the database, while 8 handles the GUI.

2. Modules

Before use of the system, it should know which module should be loaded. Each module has its own parsers and characteristics. E.g. a Word-document should be parsed different then an Excel-document. This module-class is based on the singleton pattern, so it is constructed via an abstract method which automatically loads the class and the classes needed for the parser.

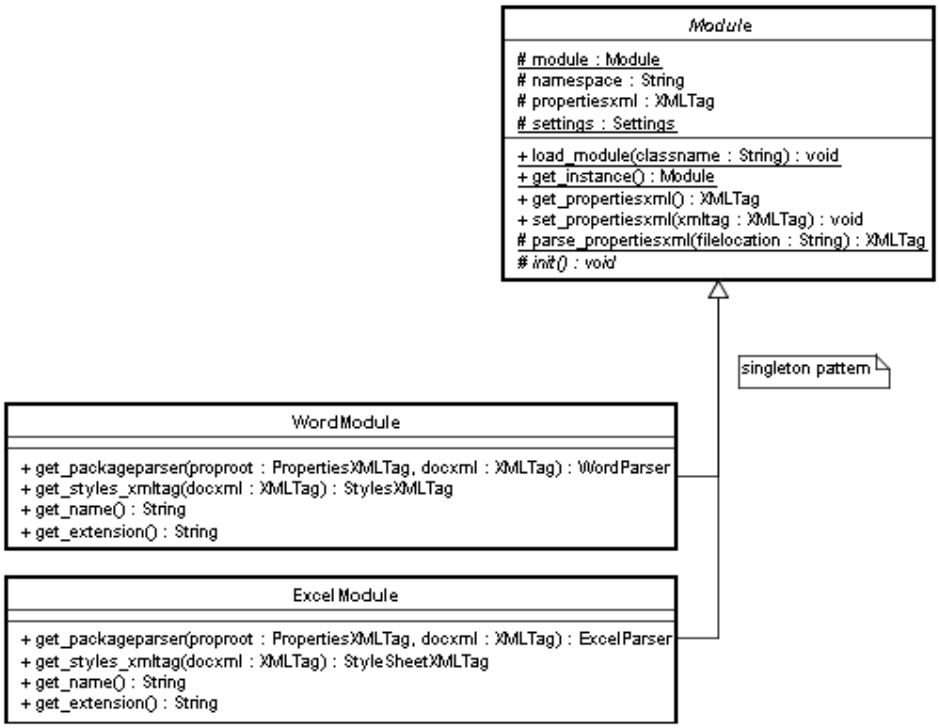


Figure 9: Class diagram: the module classes

3. Parsers

The system is built on parser technology. Many parsers are used for different tasks. To keep the bunch of parsers ordered some abstract classes are made. Every parser in the system is an OSAPParser. The parsers dealing with DocumentElements (like the level2 parsers and document sweepers) are always DocumentElementParsers. The upper parser is the MainParser. This calls the Module-class to get the PackageParser witch contains a pipeline of DocumentElementParsers used by the specific module.

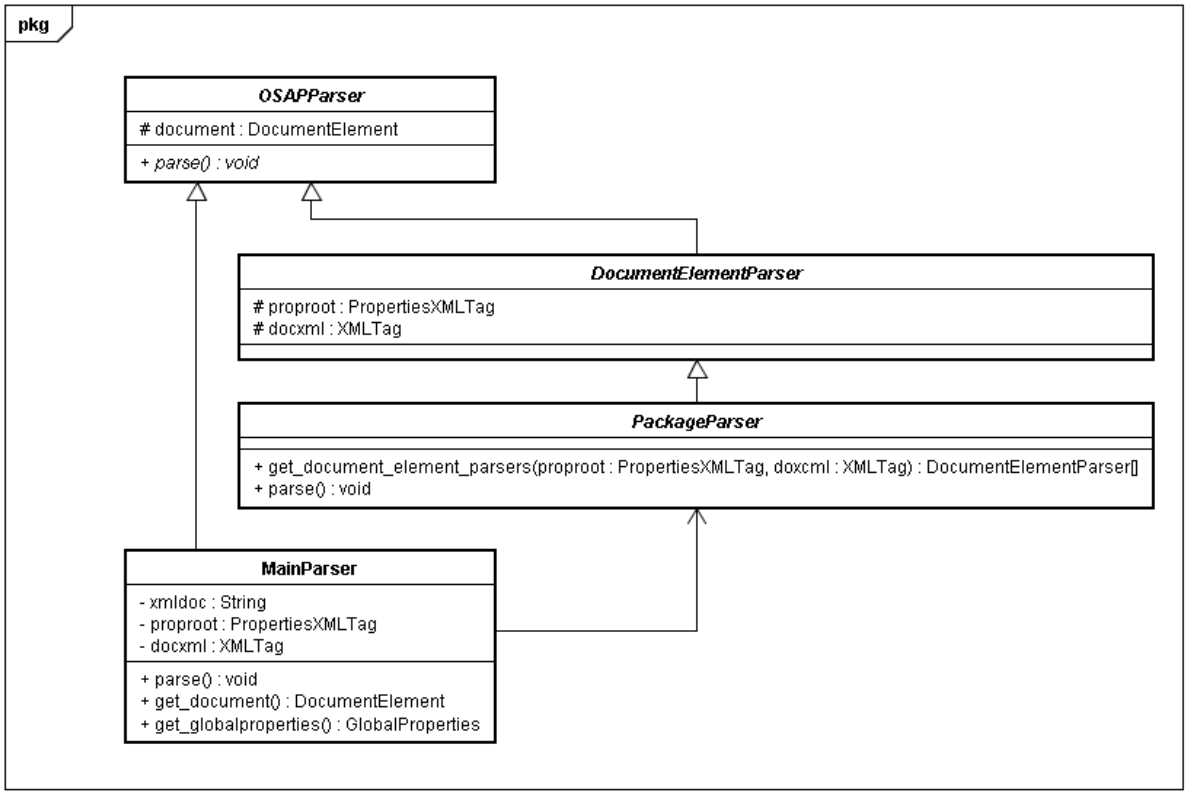


Figure 10: Class diagram: Parser structure tree

4. MainParser

The core of the system is the MainParser. This parser has the task to parse a document saved by a user in OOXML-format to the DocumentElement structure. The input of the parser is the contents of the file in String format and the output is a DocumentElement tree. The parser uses different other parsers to function correctly. In this paragraph the top level of document parsing has been worked out. In the next paragraphs the used parsers are described in detail.

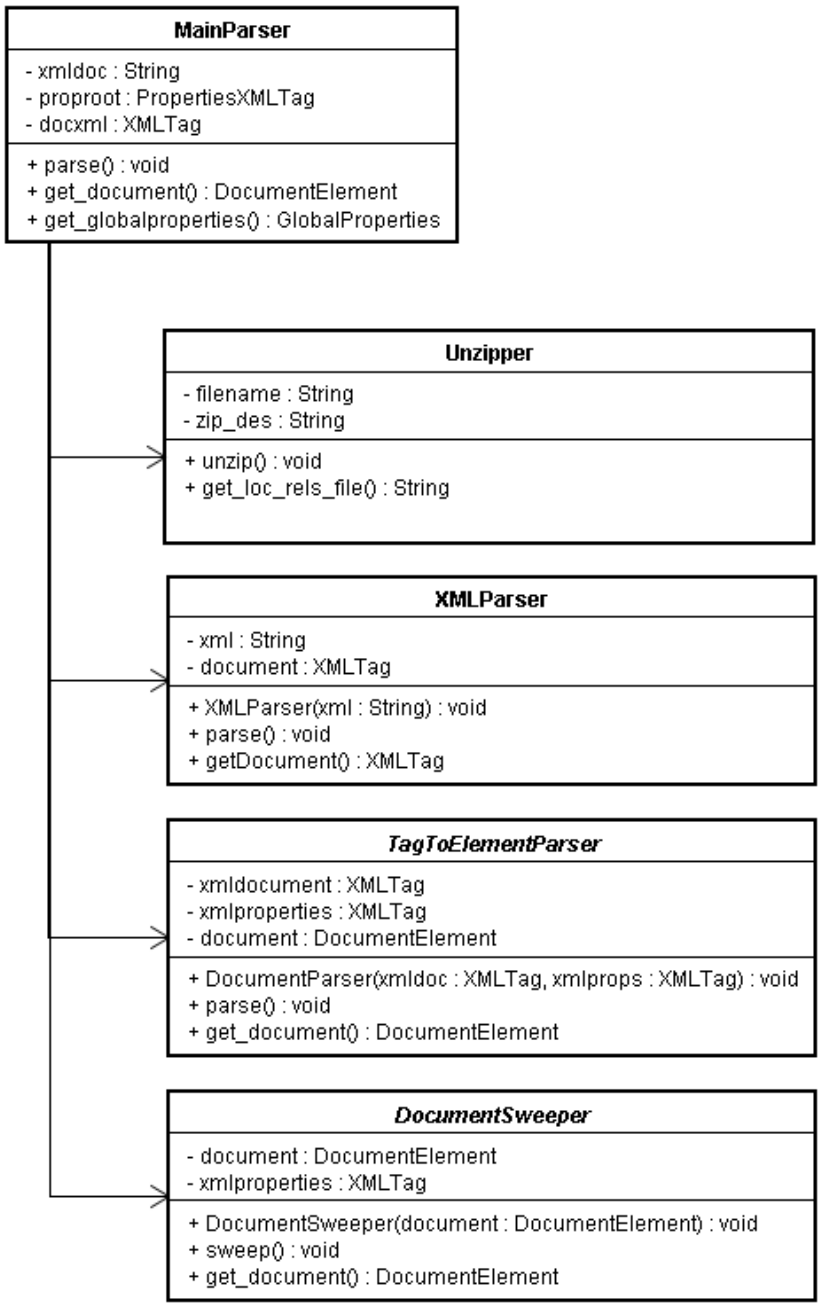


Figure 11: *Class diagram: the highest level of document parsing for a Word document*

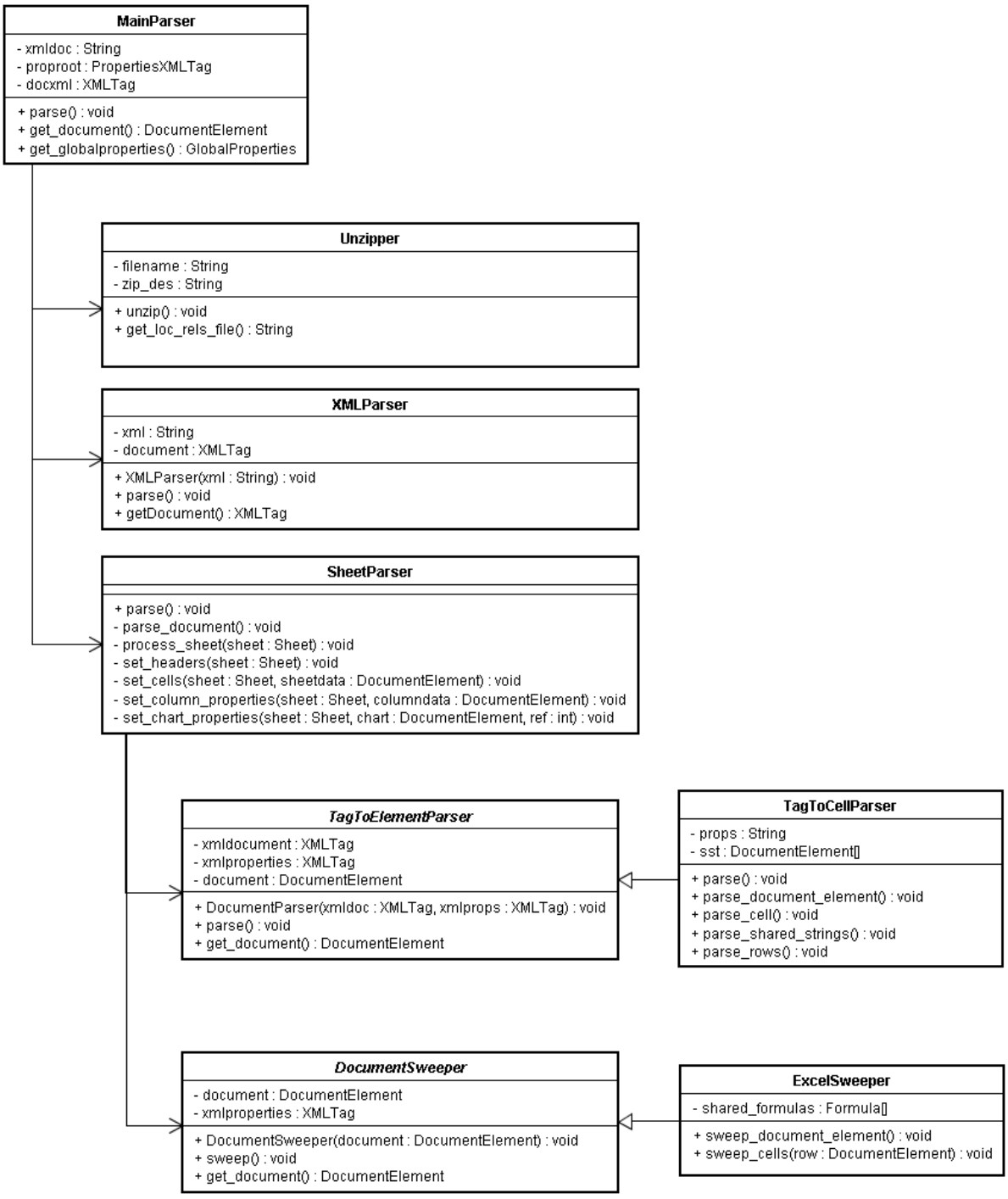


Figure 12: *Class diagram: the highest level of document parsing for a Excel document*

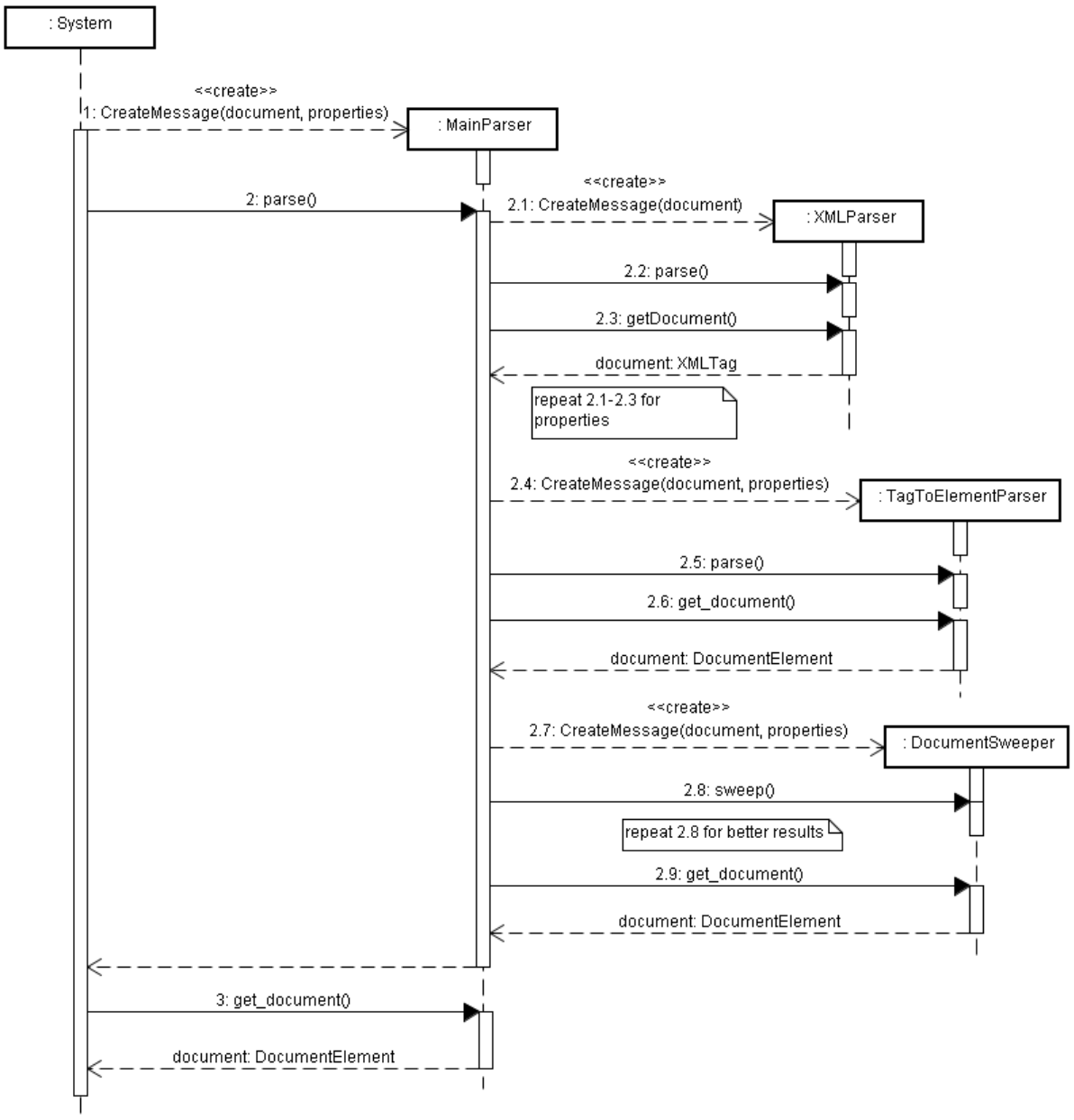


Figure 13: *Sequence diagram: parsing a Word document (highest level)*

5. XMLParser

The used XMLParser is based on the XML Parser of Adam A. Flynn[7] and has been extended wherever needed to let the parser work properly in the given context. The parser's objective is to parse an xml file in String format to an XMLTag tree format.

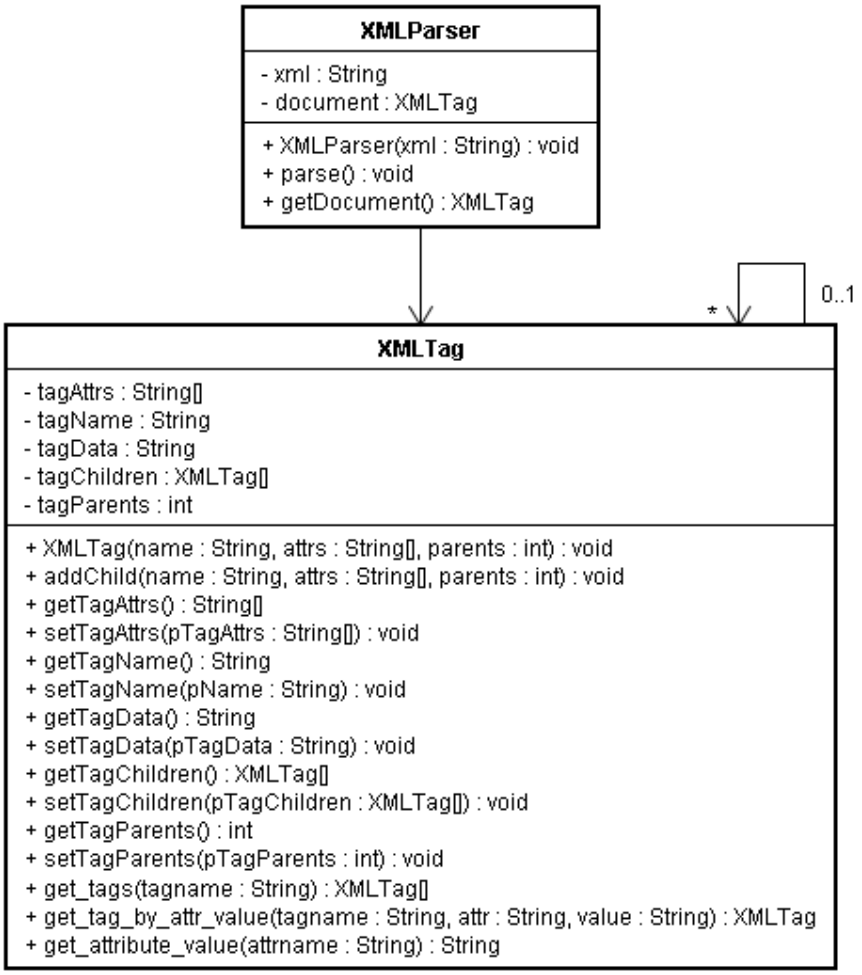


Figure 14: Class diagram: XMLParser

[7] As seen in http://www.criticaldevelopment.net/xml/. The version integrated into the system is 1.2.0 (2006) and is published under the terms of the GNU General Public License as published by the Free Software Foundation.

5.1 The iXMLTag interface

The XMLTag is used for many different kinds of XML files and parts of XML files. For some of these XMLTags special search functions are needed. Because these functions are not applicable to all XMLTags, the functions are put into different classes. The usual approach of extending the XMLTag at compile time is not feasible in this context. The XMLParser does not know which (sub-) classes to create and adapting the XMLParser – which has already been implemented by Flinn – will result in a complete other, far more complex, parser.

Extending the XMLTag at run time is a better solution. The XMLParser does not care about the extra functionality and the parsers who need the extra functionality can extend the XMLTag at runtime to add these functions. The decorator pattern[8] provides a great infrastructure to add new behavior to an XMLTag.

There are two XMLTags that are extended. The StylesXMLTag can be applied to the <w:styles> section of a Word document. And the PropertiesXMLTag provides an additional function for the <documentelements> section of the properties xml file.

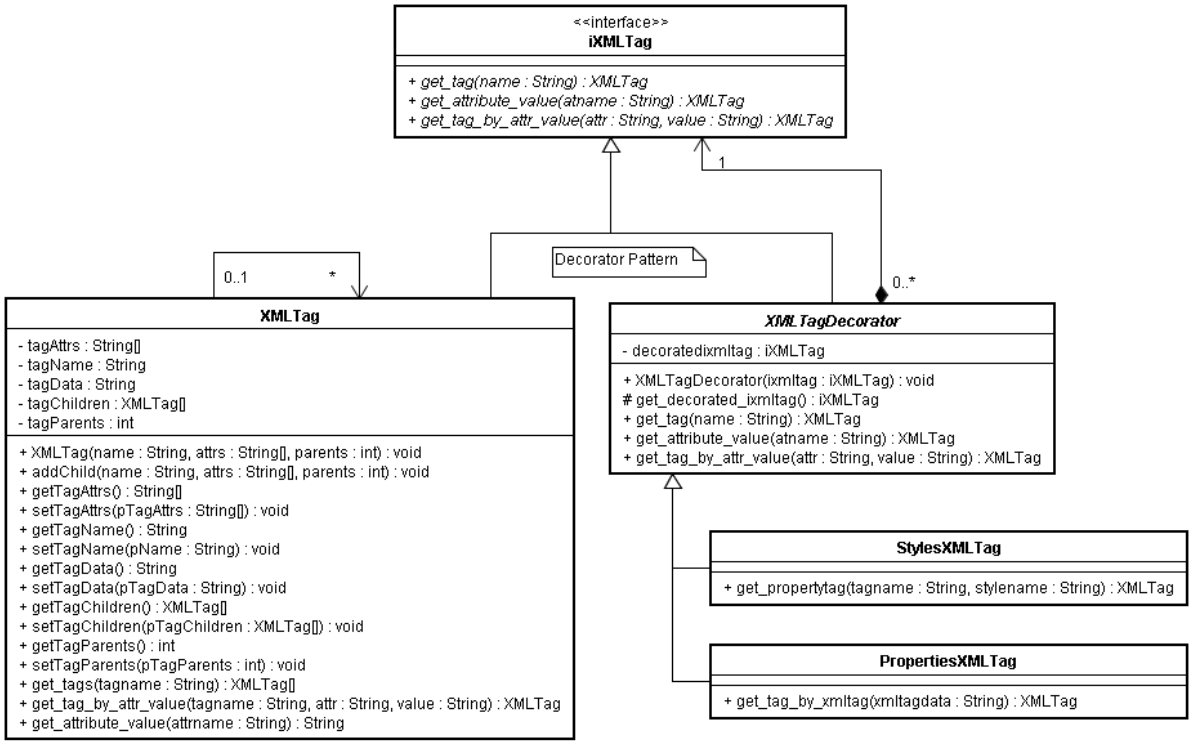


Figure 15: Class diagram: iXMLTag implementing the Decorator Pattern

[8] see for the pattern description: http://en.wikipedia.org/wiki/Decorator_pattern

6. TagToElementParser

The second level of parsing is the parsing from an XMLTag tree to a DocumentElement tree. This is the objective of the TagToElementParser.

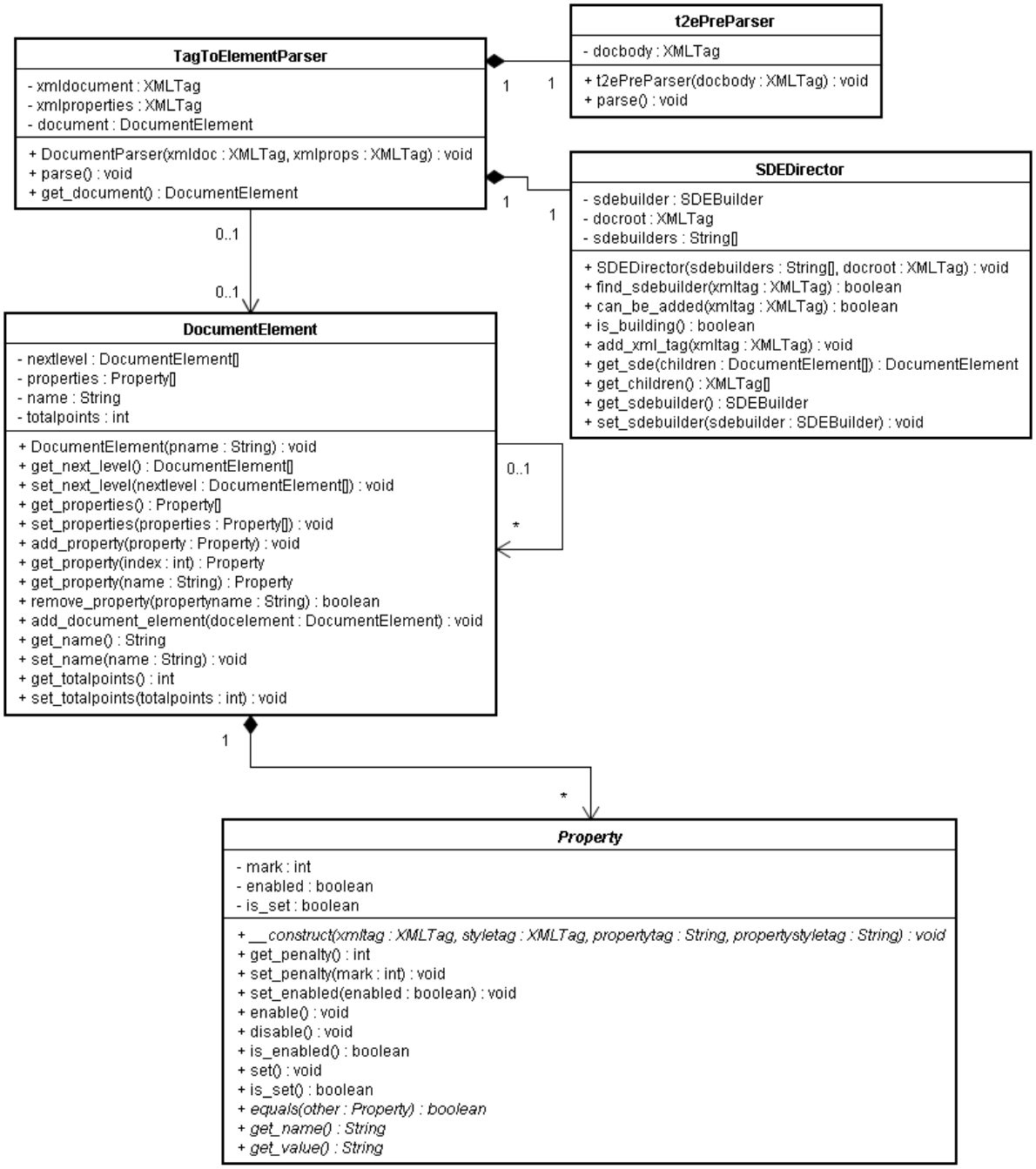


Figure 16: Class diagram: highest level TagToElementParser

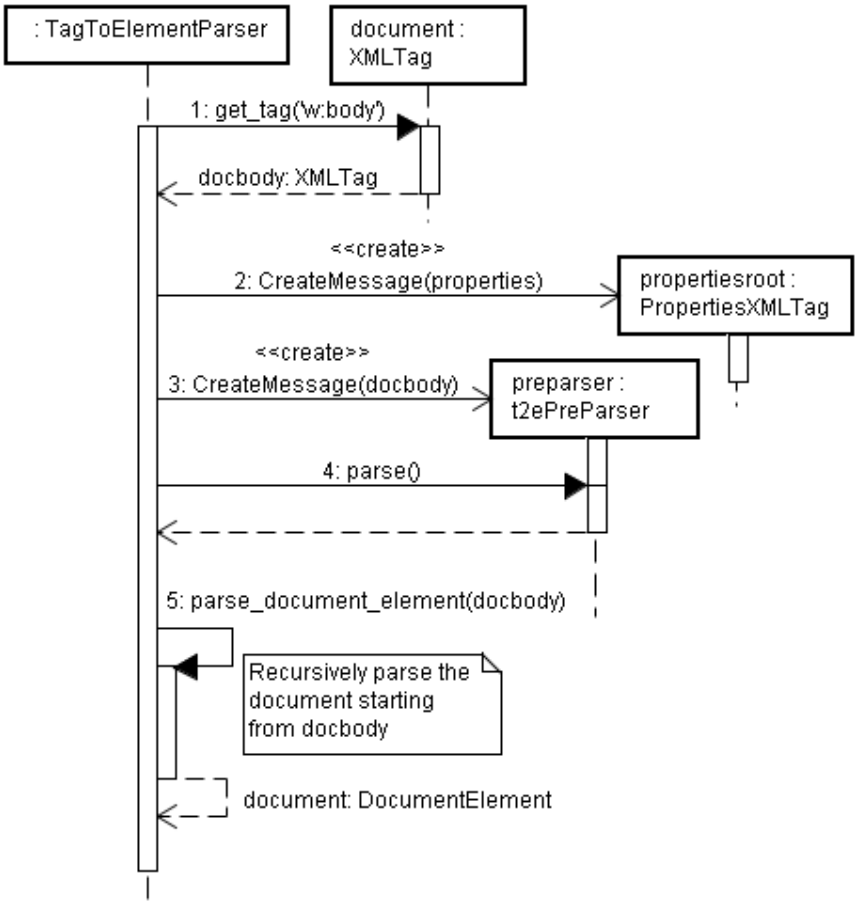


Figure 17: Sequence diagram: tag to element parsing (high level)

6.1 T2ePreParser

There are some differences between the way Microsoft Word and Open Office save documents in xml format. One of the major differences is the introduction of wx (word auxiliary) elements in the Microsoft version. Some of these elements (especially the 'wx:sect' and 'wx:sub-section' tags) makes the TagToElementParser malfunction, without adding relevant information to the system. A pre-parser has been added to remove these elements, before the actually parsing starts. After this point, both Microsoft Word and Open Office can be parsed in the same way[9].

[9] Mention that Open Office xml files does not support the complete functionality of Microsoft Word ones and are therefore not fully supported by the system.

6.2 Parsing DocumentElements

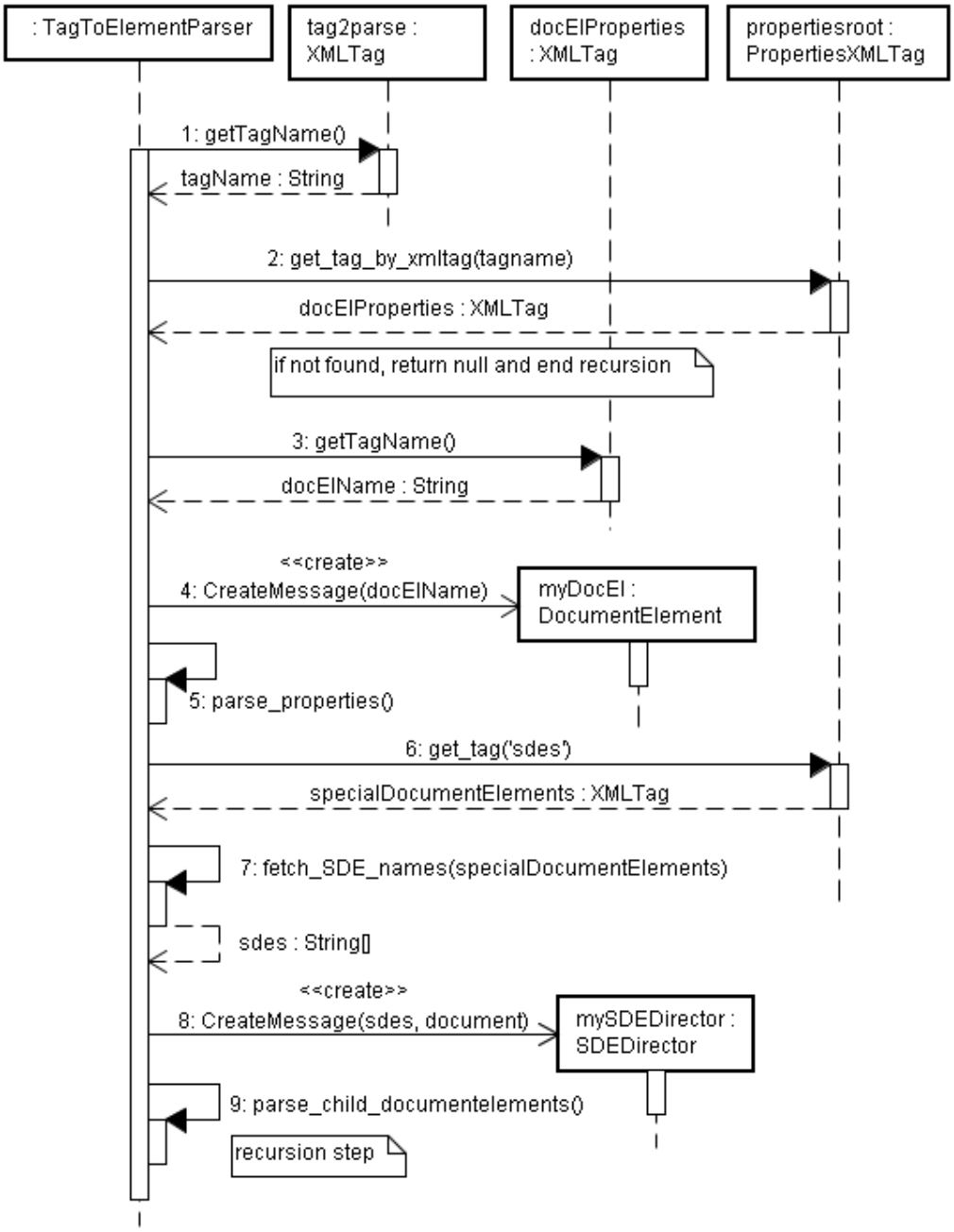


Figure 18: Sequence diagram: parse_document_element function in Figure 17. This is a more detailed level of the recursive step in parsing document elements.

6.3 Special DocumentElements

In this and the next paragraph the final recursion step (parse_child_documentelements function in Figure 18) is described. To parse child DocumentElements, it is important to know that there are two different types of child DocumentElements:

- Normal DocumentElement: This DocumentElement can be made by following the hierarchy of the XMLTag tree. For example: the current element is a paragraph. In the next level of the XMLTag a ‘w:r’-tag is found. The parser searches in the properties xml file and finds the ‘text’ element, that can be constructed out of this tag.
- Special DocumentElement (SDE): The normal hierarchy cannot be followed. For example: a list is not a special tag in the XMLTag hierarchy. Instead of this, a paragraph has a property set, in which is described that the paragraph is a member of a list. So, by only looking to the ‘w:p’ paragraph-tag, the system does not know if it is a list, or a normal paragraph. A SDE can have additional functionality to be parsed correctly.

Characteristics of Special DocumentElements

In the current parser there are two types of Special DocumentElements:

- Skip level SDE’s: In normal cases the parser will try to parse the direct children of an element. If the child is not described in the properties xml file, it will be ignored. But sometimes a grandchild is important. But parsing the child is still not necessary. A SDE could help here to skip the child level and parse the grandchild.
- Add level SDE’s: If the Word XML format does not provide a level, that is important, an extra level can be added. This is the case with a list. In XML format a list of five items is showed as five paragraphs. In this case an extra level can be added, like a list element, with five paragraphs as children, which can be parsed in a normal way.

To make this possible, the parser offers functionality to deal with the next characteristics of a Special DocumentElement:

- A SDE can be parsed in his own way. The parser will give the SDE parser all the data needed to do this.
- A SDE can have children on his own, which can be parsed in a normal way or also as a SDE. The parser offers this functionality by asking the SDE parser for his children to be parsed.
- A child SDE can be constructed out of different children of the parent. A SDE parser does not have to parse the SDE immediately. It also can buffer the children in a list, and parse the SDE in a later stage.

6.4 Parsing Child DocumentElements

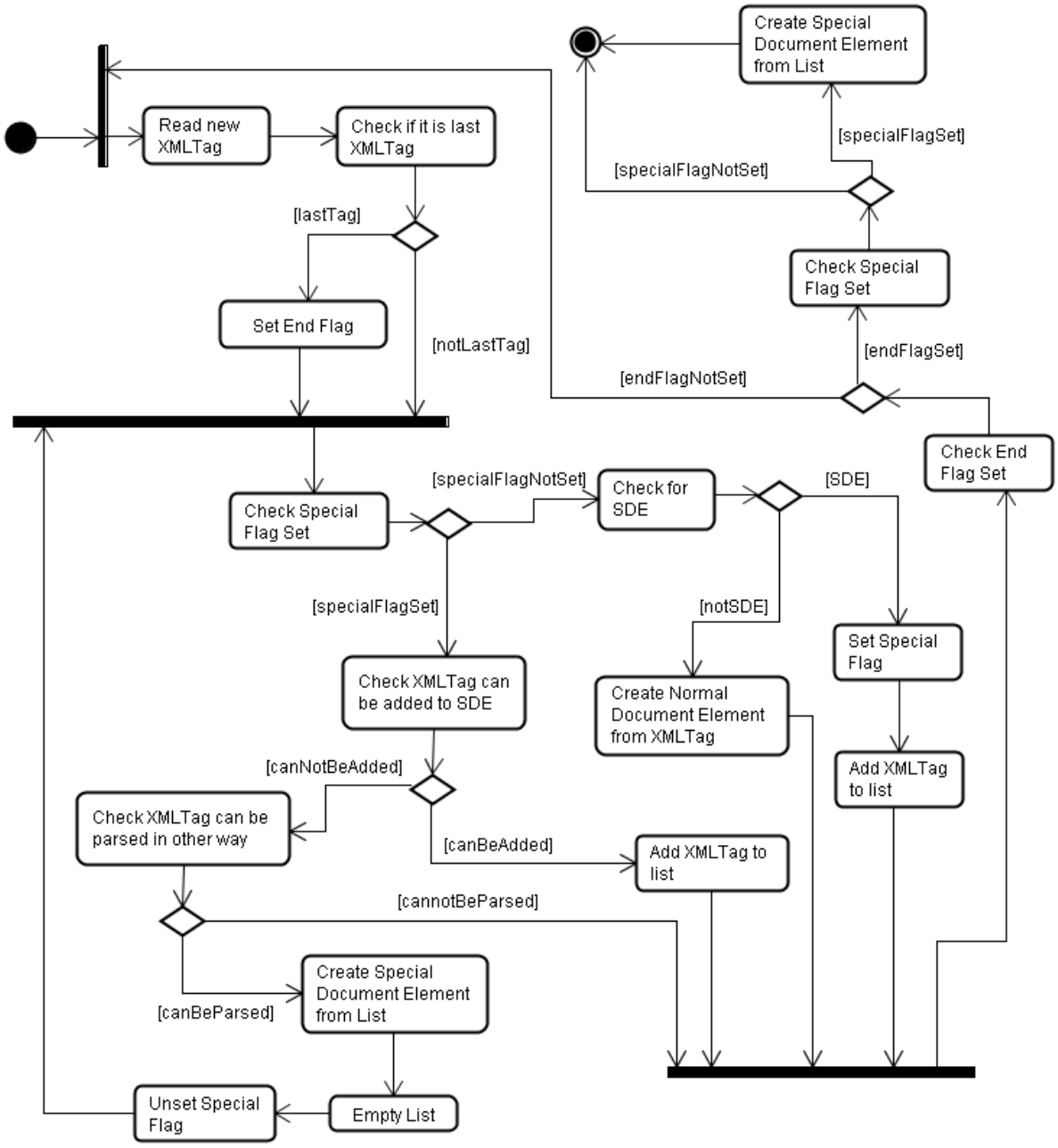


Figure 19: Activity diagram: parsing child DocumentElements

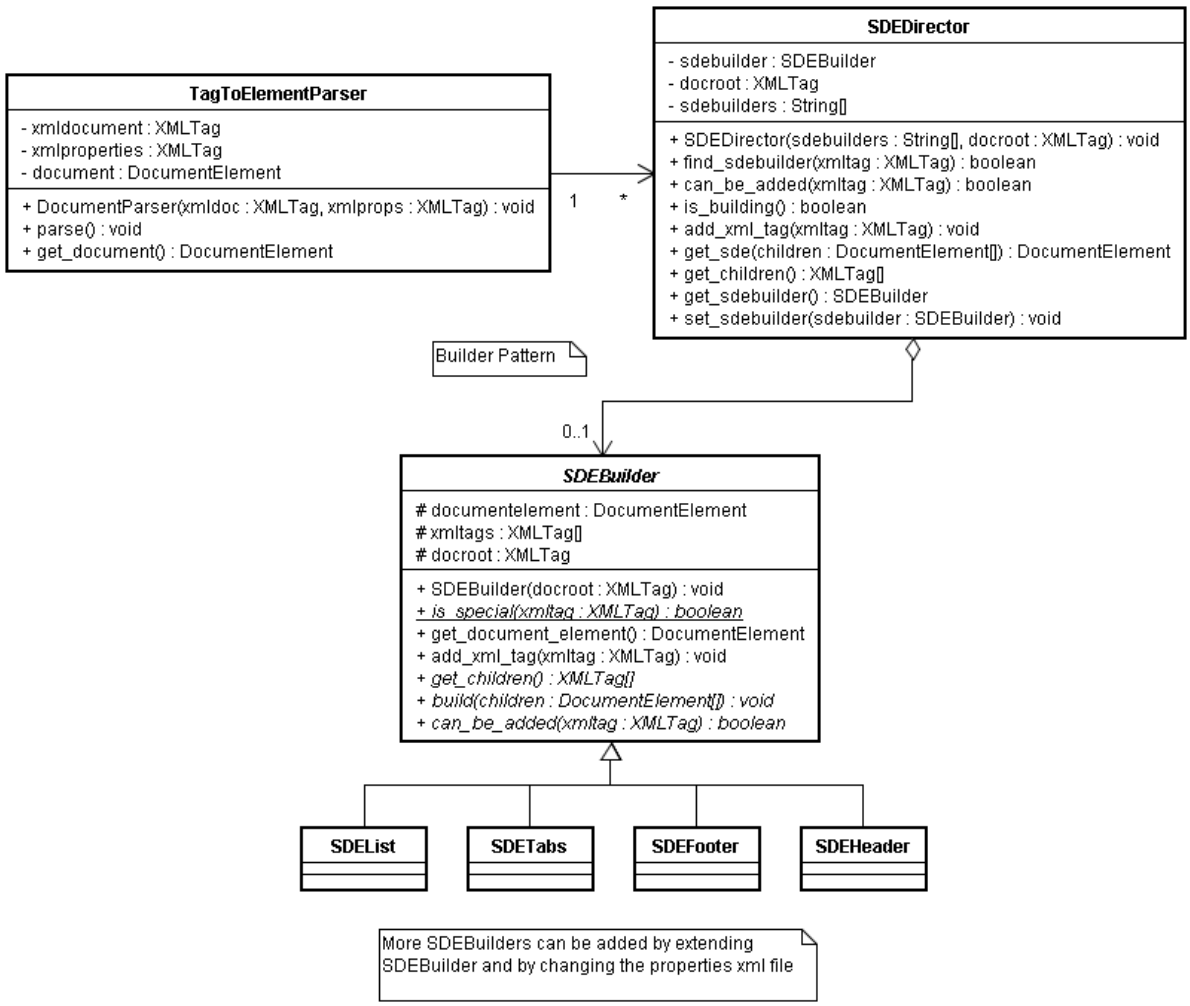


Figure 20: Class diagram: Special DocumentElement subsystem.[10]

[10] These classes are implementing the Builder Pattern. For more information about this pattern see: en.wikipedia.org/wiki/Builder_pattern

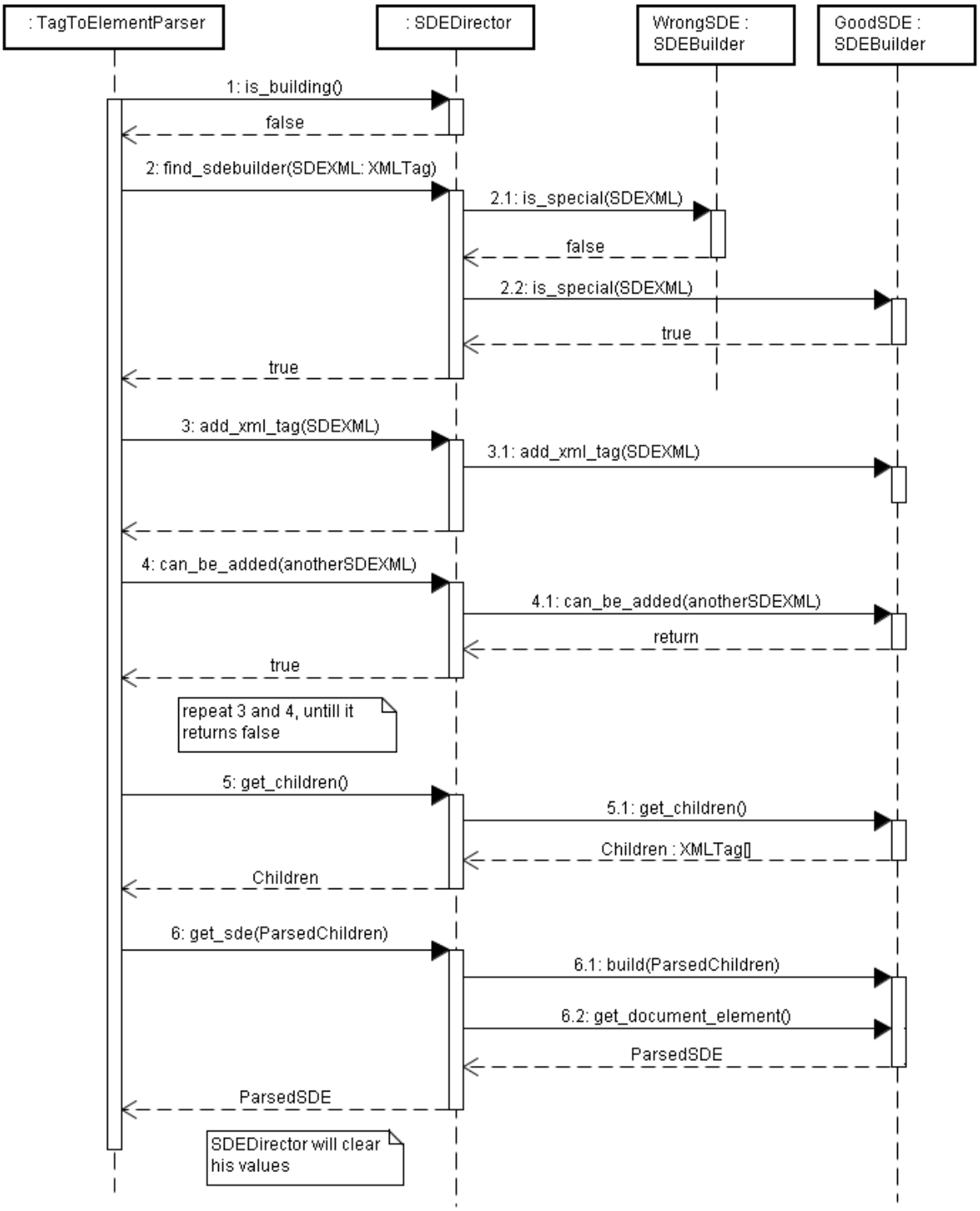


Figure 21: Sequence diagram: building Special DocumentElement

6.5 Properties

The last important task of the TagToElementParser is to parse the properties and assign them to a DocumentElement (see the parse_properties function in Figure 18.) First a description of a property should be given and the associated diagrams.

In all cases a property can be made by extending the abstract Property class and implementing the required methods. After editing the properties xml file, the parser has all the information required to parse the new property. The structure of the Property class has been given below.

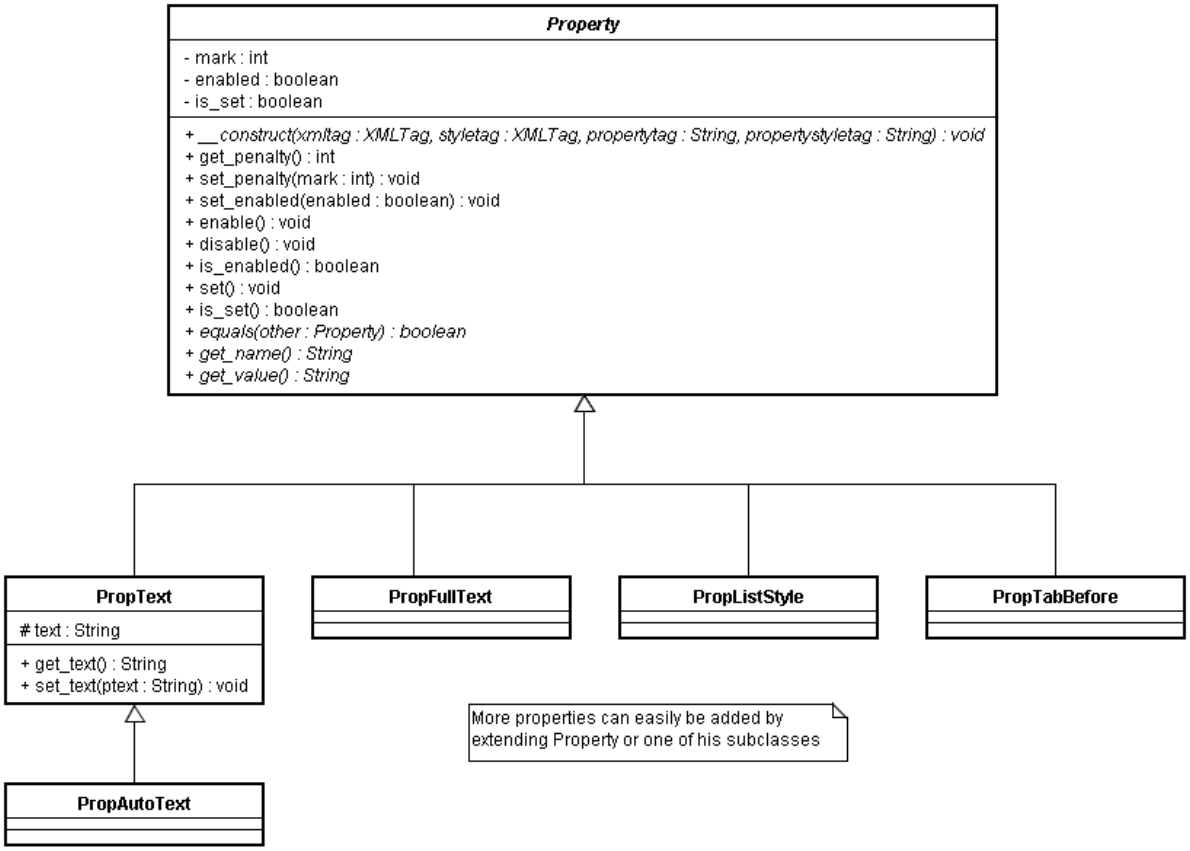


Figure 22: Class diagram: the normal Property hierarchy

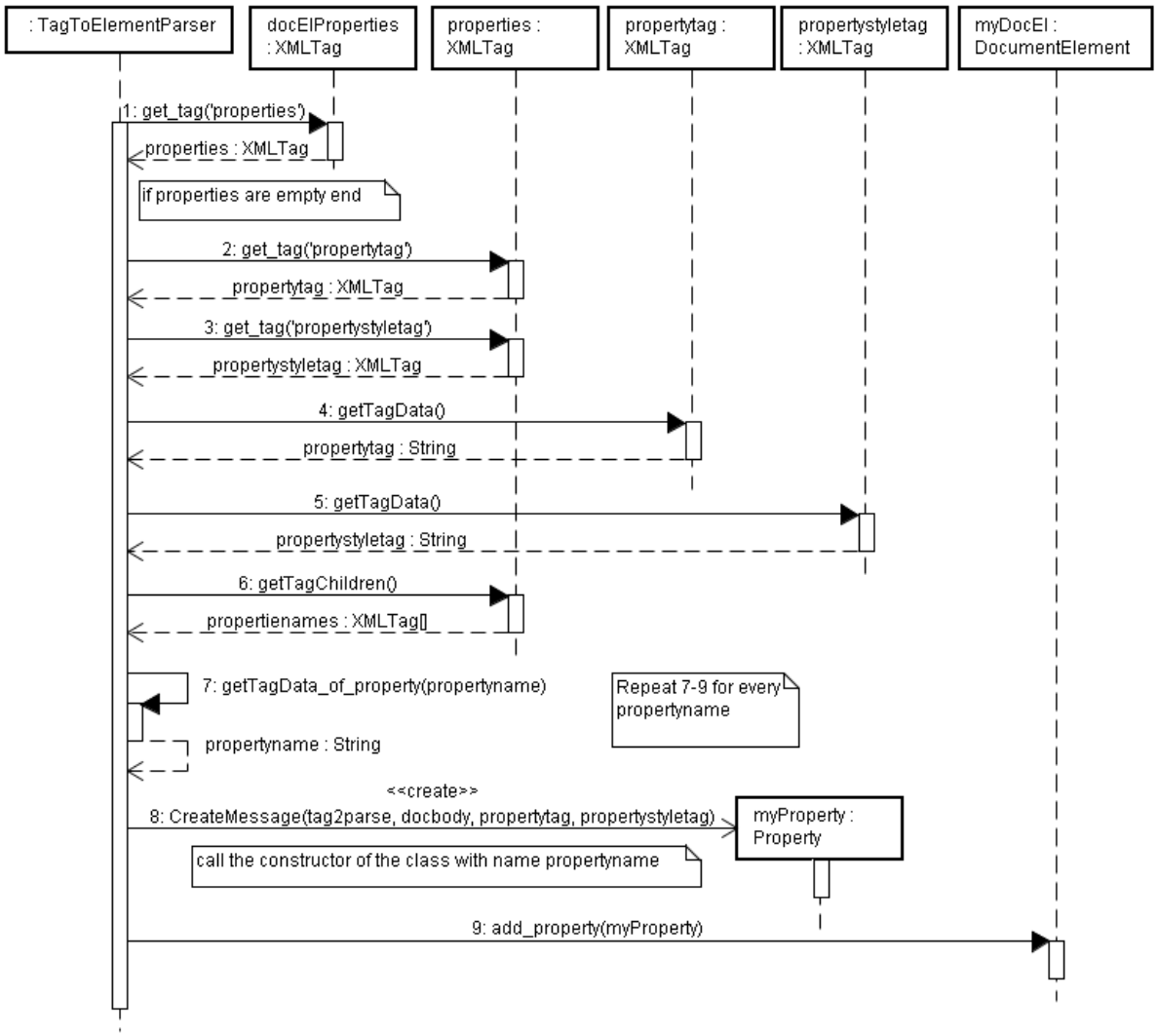


Figure 23: Sequence diagram: Parsing properties (parse_properties in Figure 18)

Styleproperties

Although the normal property hierarchy offers all the functionality required to add properties, many properties turned out to be very similar. The main two characteristics of most of the properties are:

- The data type of the property. Some properties present themselves as Booleans (like the PropBold property, which describe whether a piece of text is bold or not), others may be Strings or Integers.
- The element where the property is applied to according the Office XML standard. Some properties are paragraph properties (like justification); others are applied to characters (like font type).

For both characteristics different functionality is provided. In the perfect case each style property can choose a data type and an element and use the functionality of both. The main problem is that most languages do not support multiple inheritances. Therefore simple extension is not feasible.

The Player-Role Pattern[11] provides a solution to this problem.

All the StyleProperties can now be defined in the properties XML file. The StylePropertiesClassLoader can load the properties and add them to the property-hierarchy.

Figure 24: Class Diagram: StylePropertiesClassLoader

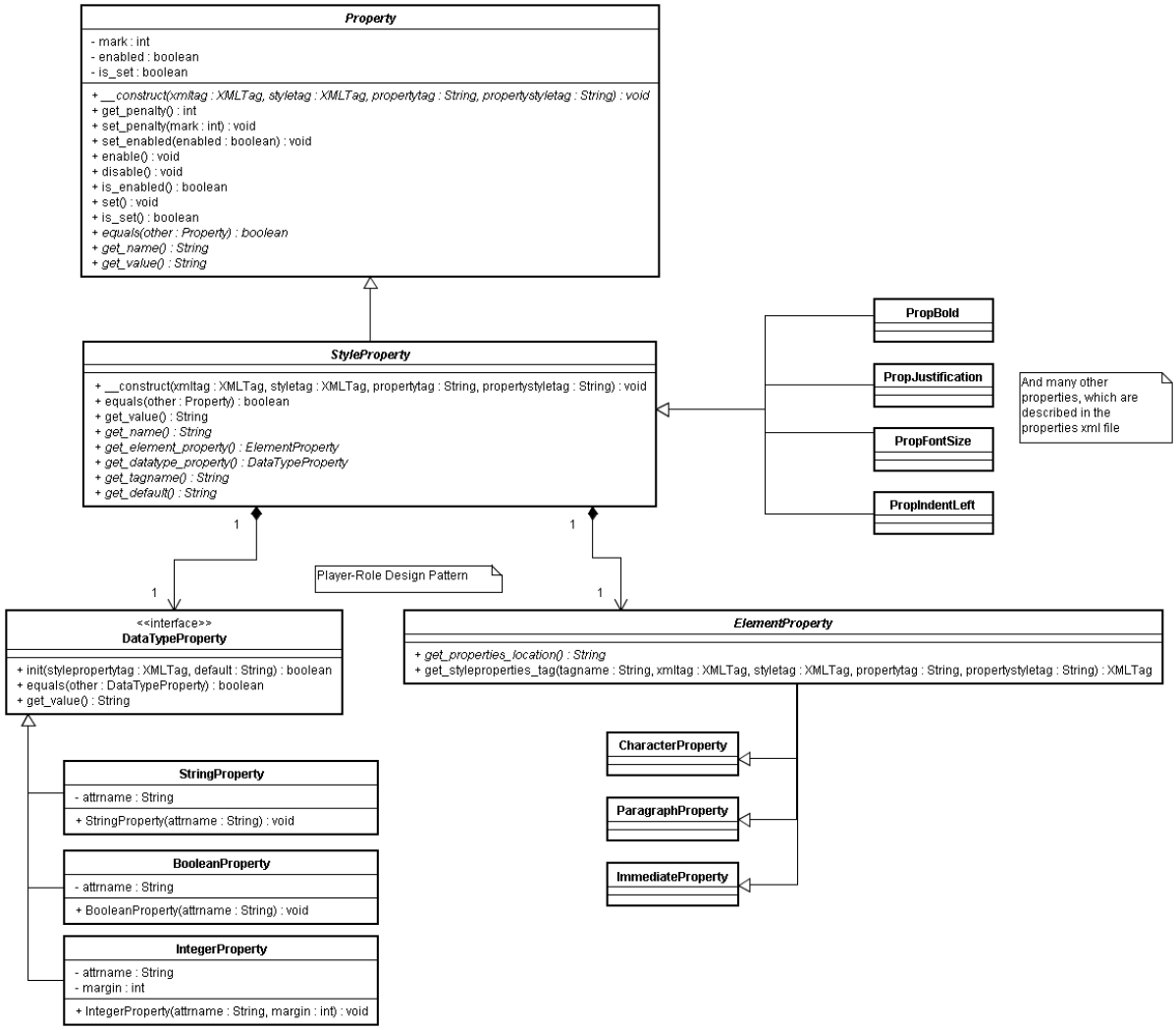


Figure 25: Class diagram: styleproperties

[11] Lethbridge, T.C., Laganière, R.: Object Oriented Software Engineering second edition, 2005, p. 228

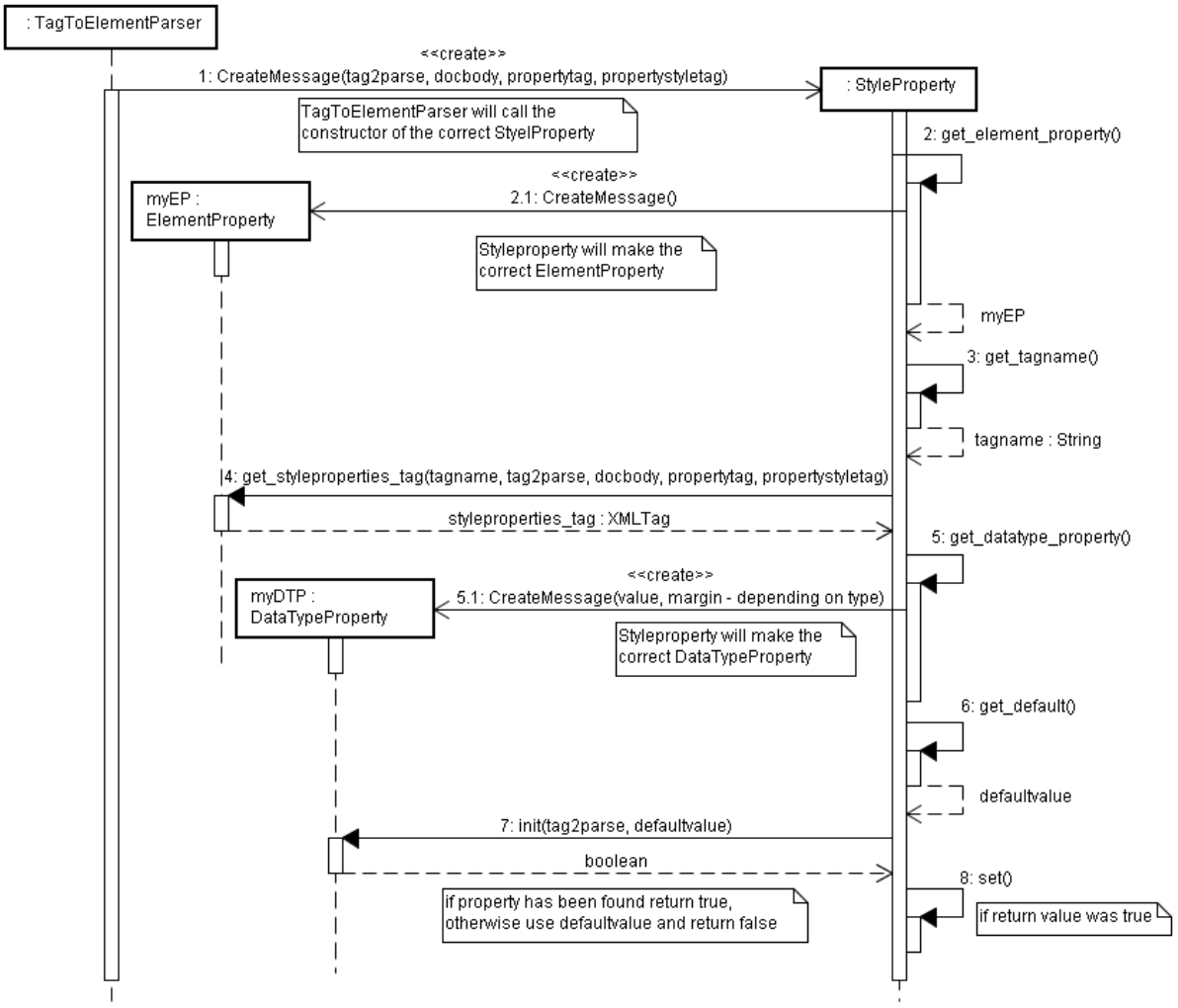


Figure 26: Sequence diagram: parsing styleproperties

7. Documentsweeper

The last level of parsing is the document sweeping. Some additional rules are added to the system to remove unnecessary data. Some rules are described in chapter 5.2.4 . For each rule a Rule class has been added. Rule 1 or A is implemented by DeleteChildPropRule, 2 (B) by DeleteParentPropRule, 3 (C) by TextRule and the last one (4/D) by NullPropRule.

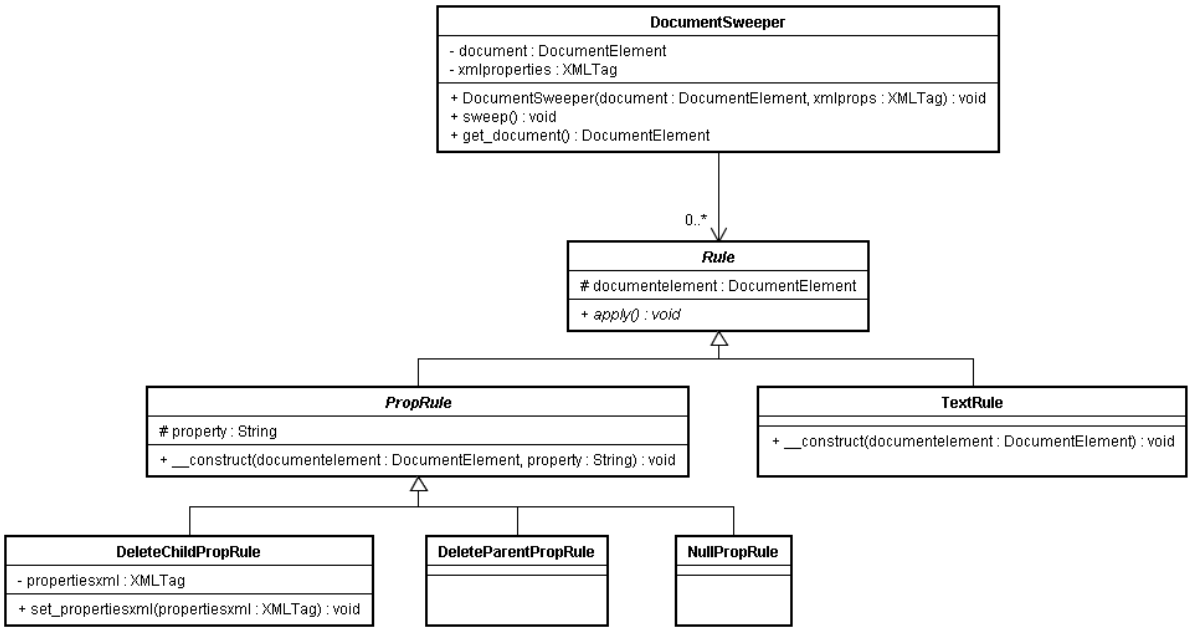


Figure 27: Class diagram: DocumentSweeper

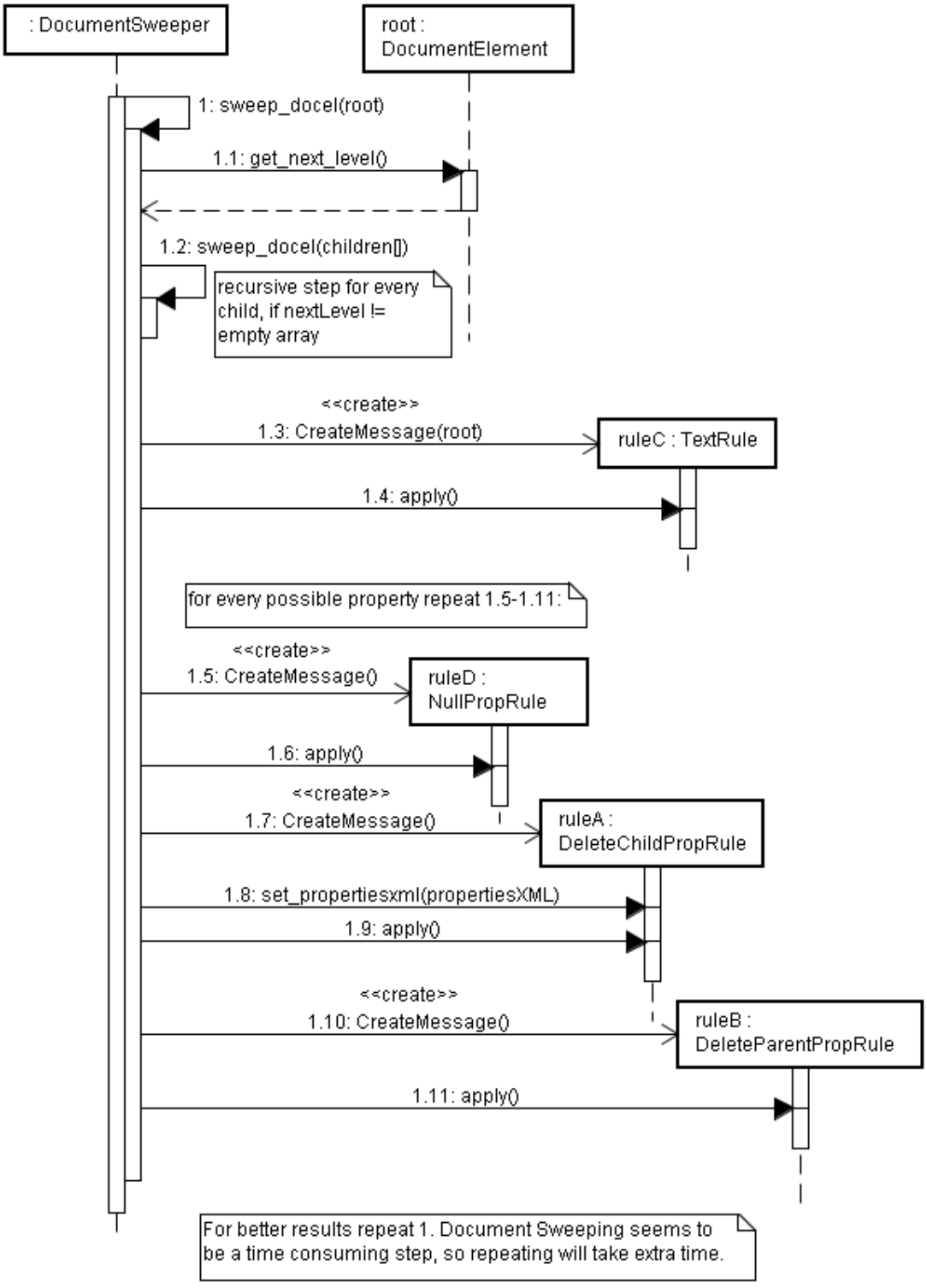


Figure 28: Sequence diagram: Document Sweeping

8. Excel Structure

To implement the Excel system some additional structures and parsers are built. The main difference between Excel and Word/Powerpoint is that the last two are based on a tree structure, while Excel is working with a matrix. We still can use the DocumentElementStructure but below the Sheet-layer there is no longer an array of Cells, but a matrix. Also a new class: Formula has been created, which handles the formulas of Excel.

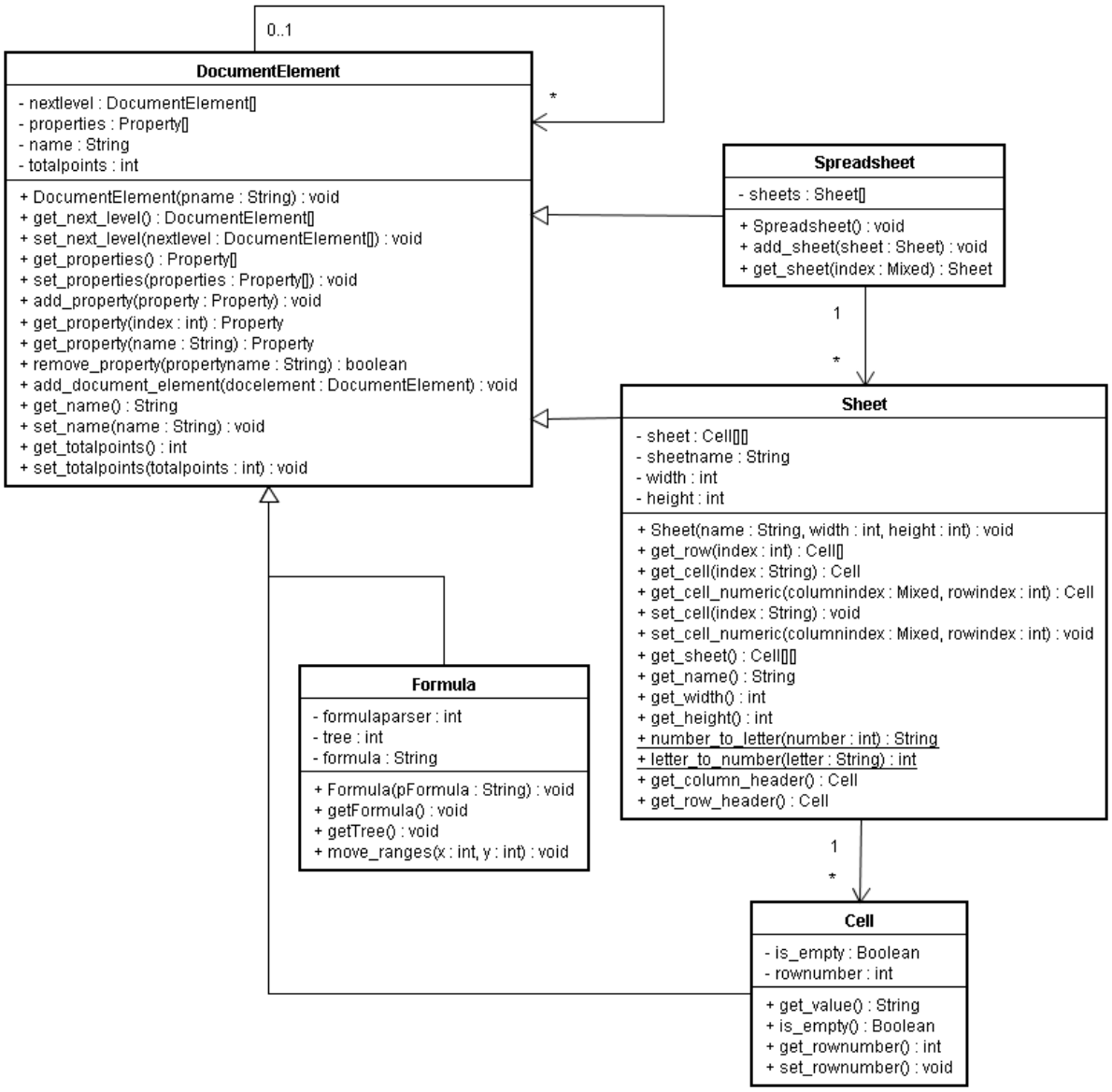


Figure 29: Class Diagram: Excel Structure

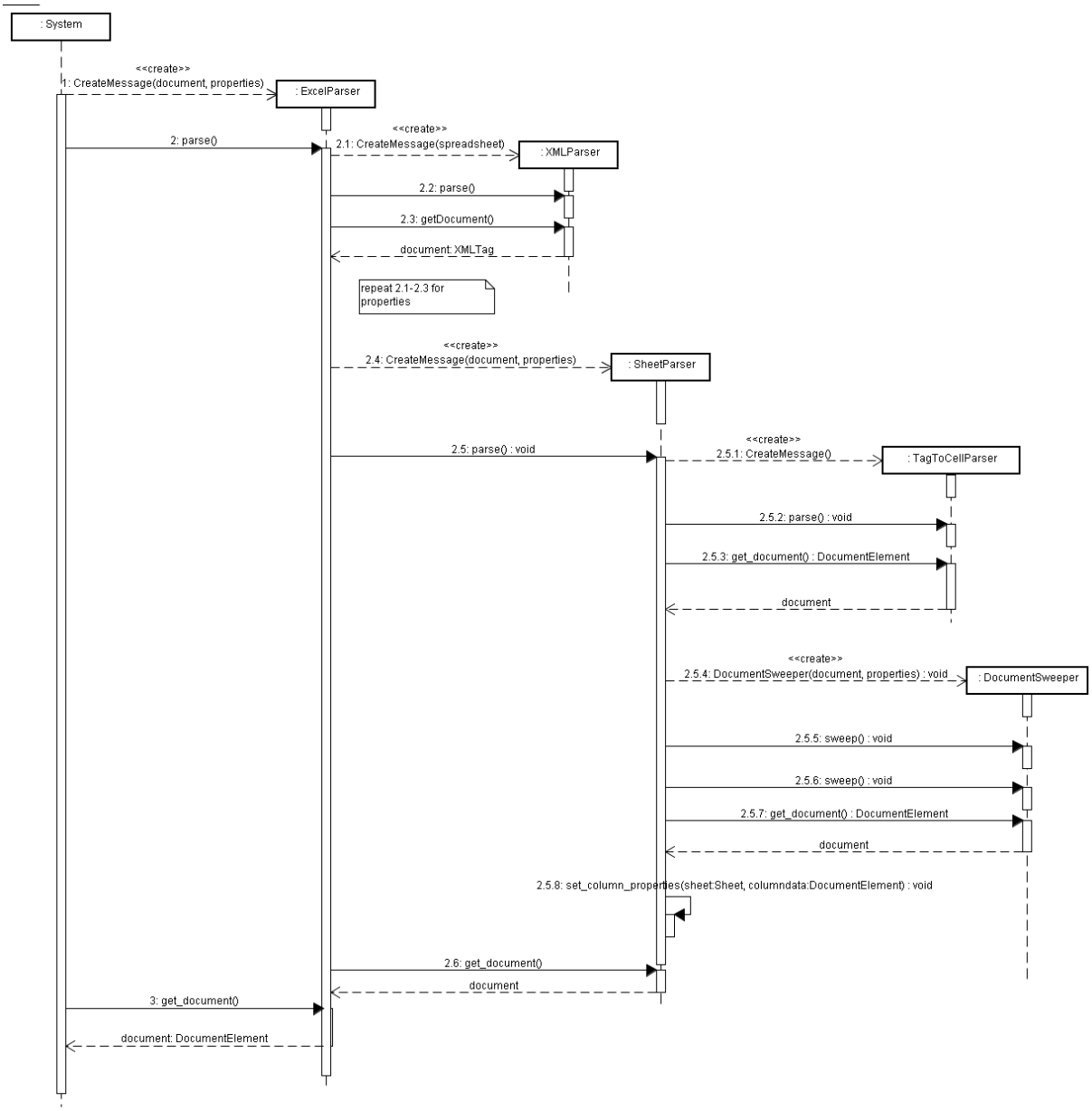


Figure 30: Sequence Diagram: Cell Parsing

9. Document Marker

The last part of the core system is the document marker. It is designed to compare two documents and give a mark to the student's document. The way of assessing documents and mark them is described in section 5.2.5 of the design chapter.

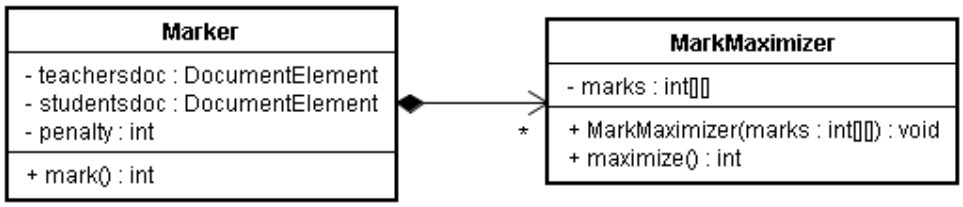


Figure 31: Class diagram: Marker

The marking algorithm starts with the teacher's documentroot and the student's documentroot and marks the document elements recursively as shown below.

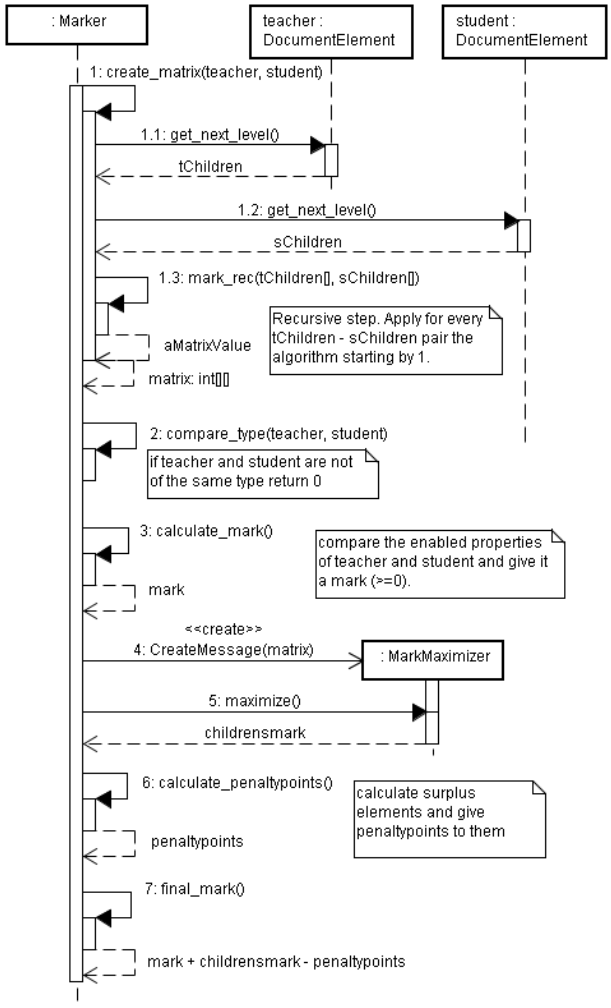


Figure 32: Sequence diagram: recursive step of marking algorithm

Markmaximizer

An important part of the marker is the Markmaximizer. Its goal is to compare two sets of document elements and find an optimal solution for the mapping of the document elements of the teacher to the student's ones. The input is a t x s matrix of integers. T is the number of teacher's document elements and s is the number of the student's ones. The markmaximizer will try to find optimal pairs of teacher's and student's document elements, corresponding to the constraints mentioned in the previous chapter (like: the ordering of document elements is important.)

10. The OSAP database / filesystem

The OSAP database consists out of a mysql database and the file system of the server. The database is used for saving the tests, the student solutions and the messages created by the OSAP system.

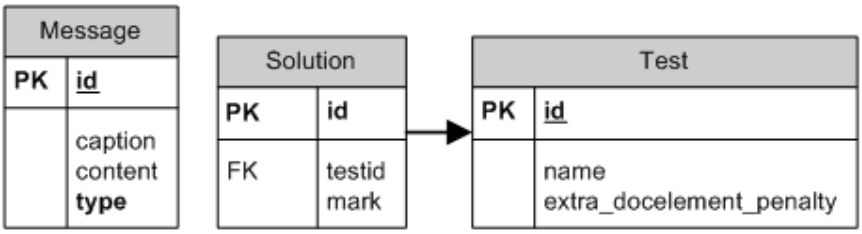


Figure 33: OSAP database diagram

After the teacher and solution documents are uploaded and parsed into document element structures. An original of the word document is backed up and the DocumentElements structure is saved as binary object file on the server's file system.

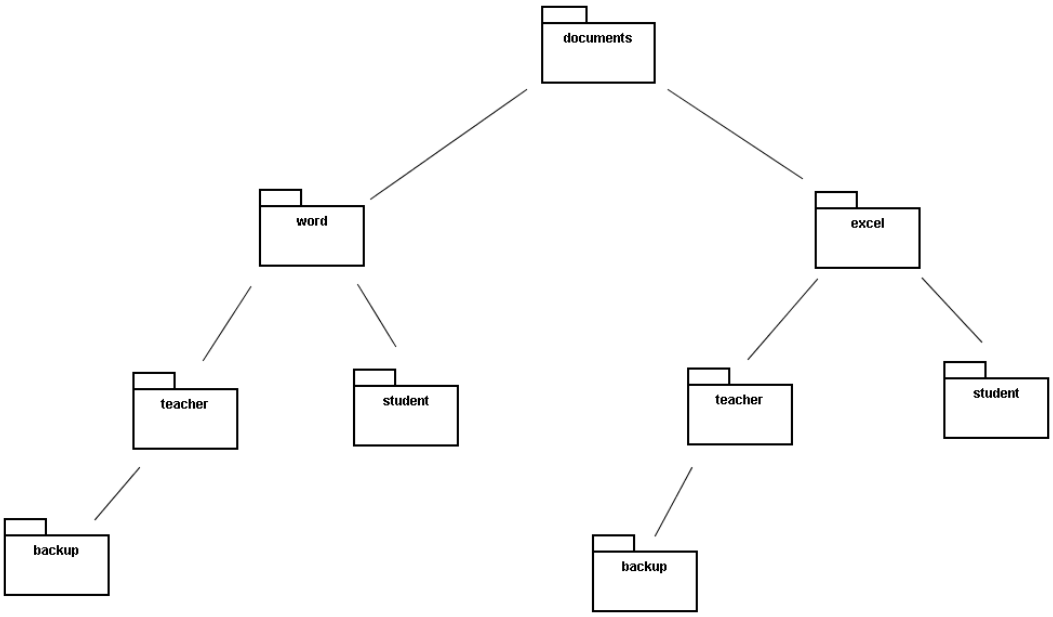


Figure 34: Directory structure of saved Documents

In the teacher and student directories the binary files and in the backup directories the original Word documents. The binary DocumentElement files are saved in the root of the 'teacher' and 'student' directories and the original documents in ooxml format in the backup directories.

The communication with the OSAP database takes place using the instances of PHP classes:

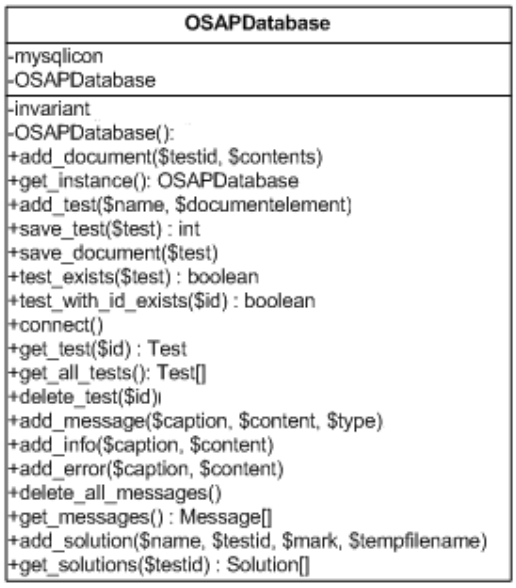


Figure 35: Represents the OSAPDatabase with functionality to modify, delete, get and add data in the database.

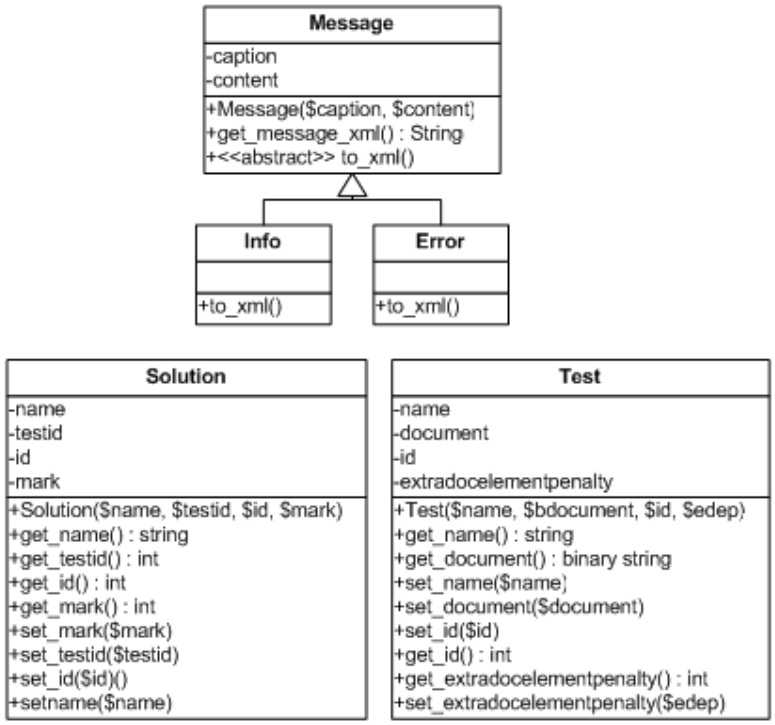


Figure 36: Represents the data structures stored in the OSAP database.

11. The Graphical User Interface

This section describes the working of the user interface of the system. First the objectives for the design of the user interface are described in section 8.1. Section 8.2 gives an overview of the chosen techniques for developing the user interface based on the objectives from section 8.1. With the choice of the techniques, the structure of the GUI design is described in 8.3.

11.1 The objectives for the GUI

- Separating data from representation
- Providing a structured view of the document structure and marking scheme
- Instinctive usability
- Fast server responses

11.2 The techniques

- PHP
    - Data manipulation
    - Communication with Mysql database
    - XML output
- XSLT (eXtensible Stylesheet Language Transformations)
    - Transforming XML into HTML
- AJAX (Asynchronous Javascript and XML)
    - Event handling
    - Providing fast interaction with the server. A page doesn't has to reload completely on every action of the user.
- CSS
    - Styling the html produced by the XSL transformation

11.3 Structure of the GUI

In the structure of the GUI there is a distinction between the output in the form of web pages and the data manipulation functionality packed in PHP classes.

Class structure of the GUI

The GUI consists out of a student side and a teacher side.

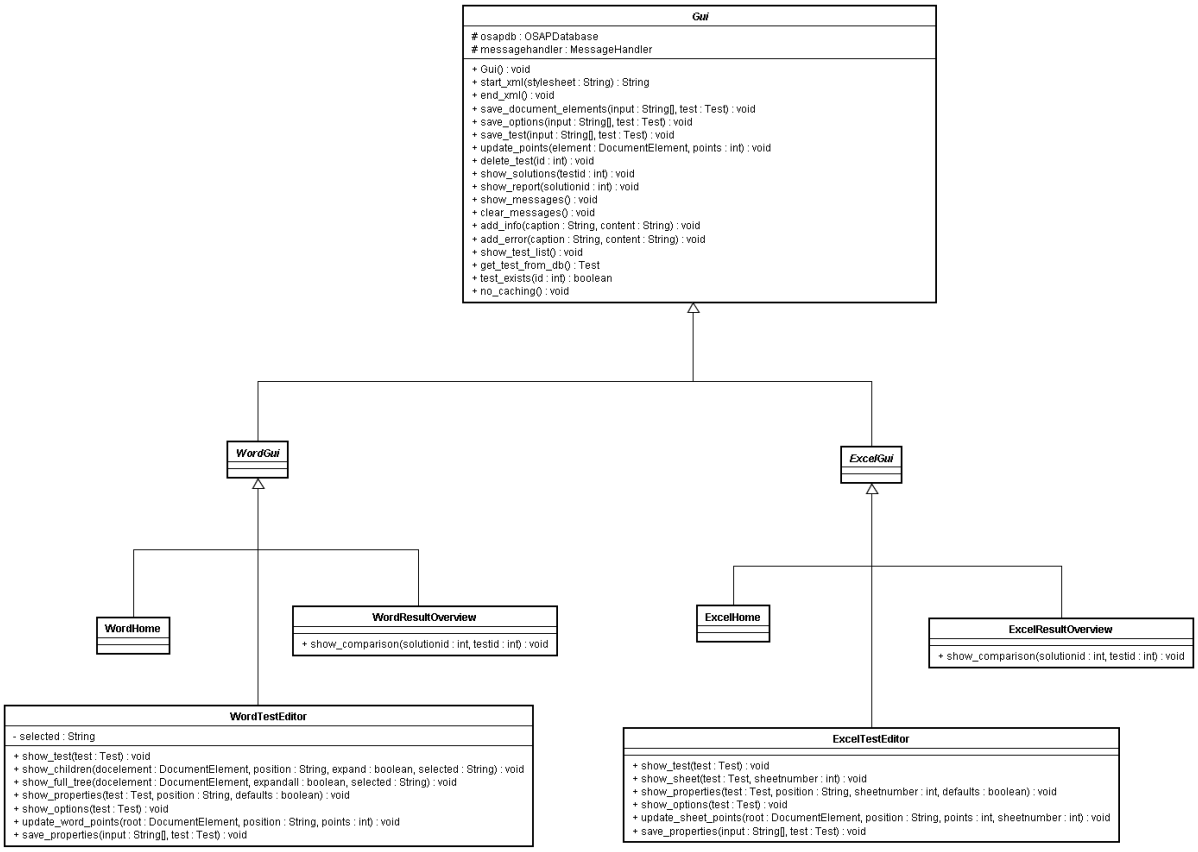


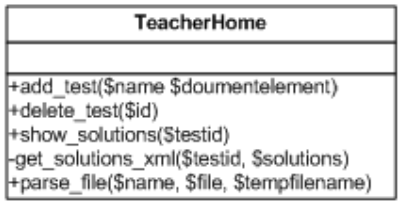
Figure 37: Class relation diagram of GUI classes

The teacher side

The teacher side contains a home, a test editor and a result overview part. Represented by the classes:

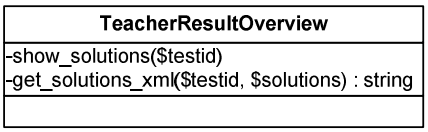
TeacherHome

Functionality for outputting a list of the current tests in xml format and functions for manipulating the tests and adding new tests.



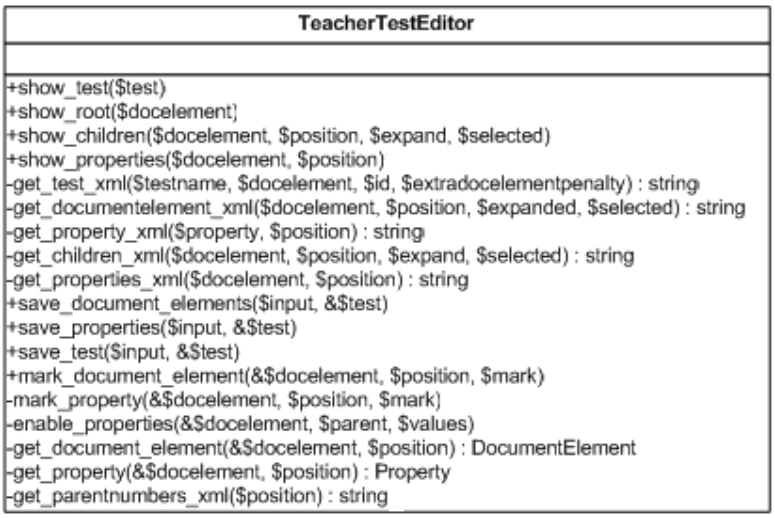
TeacherResultOverview

Functionality for outputting a list of the current student solutions of a selected test.



TeacherTestEditor

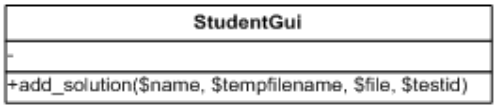
The main part of the teacher side. Functionality for outputting a test and the corresponding teacher solution document in xml format and processing the changes in a test made by the user (teacher).



The student side

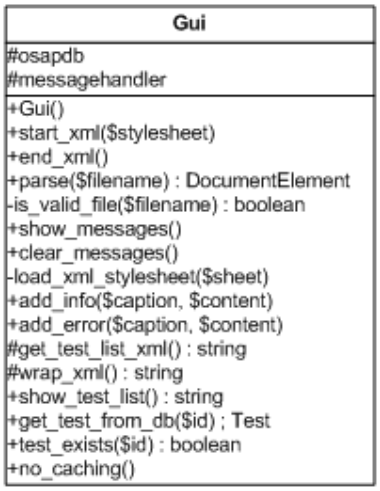
StudentGui

Functionality for uploading and marking student solutions.



Gui

The common functionality of the different GUI parts.



Page structure of the GUI

There is a distinction between normal pages and AJAX result pages.

A normal page is a PHP script which outputs XML data including a reference to a XSL file. When a user browses to a normal page the browser performs the XSLT transformation and transforms the XML to HTML including Javascript code and a reference to a CSS file. Javascript functions are called when the user interacts with a page. These functions trigger page requests to the server. Each answer from the server corresponds with an AJAX result page which after an XSL transformation can be put in a HTML container. The rest of the already loaded normal page doesn't has to reload which makes the GUI more responsive.

An AJAX result page is also a PHP script which outputs XML data, but the transformation is done with an XSLT parser build in javascript: Google AJAX XSLT[12].

Next to the Google AJAX XSLT library the GUI also uses the Sarissa library[13]. This library makes it easier to add actions to events.

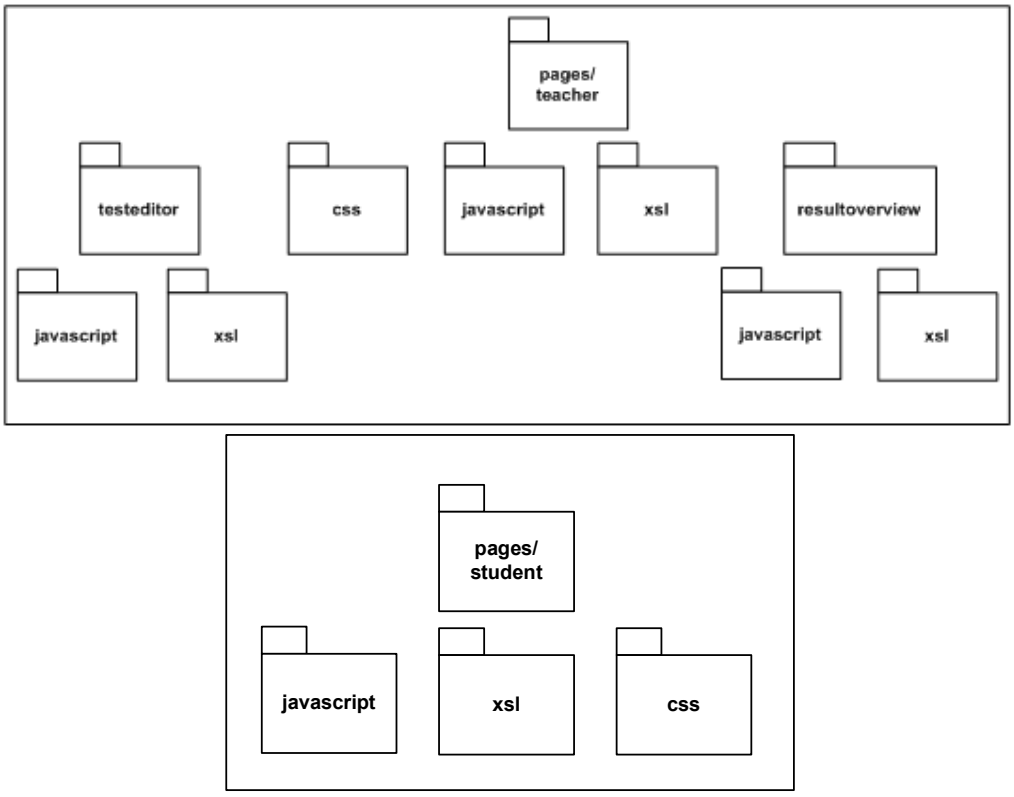


Figure 38: Directory structure of the GUI pages. The javascript files and the XSL files are separated from the PHP files.

[12] http://goog-ajaxslt.sourceforge.net/
[13] available at http://sourceforge.net/projects/sarissa

Appendix C: XML Syntax Properties File

The expert data to decide which properties are important to check on is provided to the system using a XML properties file. This file shows how the level 2 parser should work to make Office Skill Assessment possible. The file has is constructed using the following tags:

tag	parent	description
<worddocument>	none	Document's root tag
<documentelements>	<worddocument>	Document Element's root tag
<*a doc.el.name*>	<documentelements>	Document Element. The tag name will be used in the user interface.
<xmltag>	<*a doc.el.name*>	The tag used in OOXML format. The parser uses this tag to identify the Document Element in the original document.
<propertytag>	<*a doc.el.name*>	Property tag used in OOXML format.
<propertystyletag>	<*a doc.el.name*>	Property style tag used in OOXML format.
<properties>	<*a doc.el.name*>	Properties that are applied to the Document Element.
<property>	<properties>	Property that is applied to the Document Element. The data surrounded by this tag should be the class name of the property.
<styleproperties>	<worddocument>	Style Properties root tag
<styleproperty>	<styleproperties>	Style property
<classname>	<styleproperty>	Unique Class Name for style property. Class Name has to start with 'Prop'
<name>	<styleproperty>	Name of style property. This will be used in the user interface.
<xmltag>	<styleproperty>	The tag used in OOXML format. The parser uses this tag to identify the Property in the original document.
<defaultvalue>	<styleproperty>	Default value of the property. If default value is 'true' or 'false', the parser will handle it as a boolean. If there is no default value, the parser will not use a default value, but will copy the value of the parent.
<datatypeproperty>	<styleproperty>	The DataTypeProperty class name of the property.
<elementproperty>	<styleproperty>	The ElementProperty class name of the property. An attribute 'val' may be used to give the extra parameter to the constructor.

Appendix D: Definition of UCSC staff positions related to eBIT

UCSC Board of Management	Has the overall responsibility for monitoring and evaluation of the deployment of eBIT examination regulations, e-assessment policies, practices and procedures. Decision board for final approval of major changes and updates of related documents. Responsible for the overall organizational set up of the eBIT program.
Board of Study, External & Extension programs	Responsible for a periodical review (? x annually) and maintenance of e-assessment policies, practices and procedures in place. Proposes updates when needed and for instance triggered by upcoming opportunities offered by new technology trends within the global academic society or changing demands for local society. Will act as arbiter for cases of cheating and fraud and cases in which students progress is slowed down beyond their control. Reports directly to the Board of Management
eBIT project committee	Holding the day-to-day responsibility for the operational eBIT program including eBIT academic and technical support staff resources, eBIT financial, curriculum and student administration and related ICT resources installed at the eBIT labs and test centers. Reports directly to the Board of Study, External & Extension programs
eBIT course coordinator	Is assigned the overall responsibility for a specific eBIT module. It includes the development and maintenance of the syllabus, e-learning and e-assessment content (LMS supported question bank). The eBIT course coordinator will also be responsible for the final evaluation of e-assessment results of students for that specific eBIT course module through review of the e-assessment results produced by the LMS. Reports to the eBIT project committee.
eBIT project coordinator	The eBIT project coordinator's main responsibility is to facilitate the eBIT processes of e-learning and e-assessment by making the needed resources (ICT services, e-learning and e-assessment content) on behalf of the eBIT course coordinators available to the students. The eBIT project coordinator will in principle not be responsible for content matters. The eBIT project coordinator will report to the individual eBIT course coordinators (he or she is supporting) and the eBIT project committee
BIT e-assessment coordinator	Responsible to the day-to-day implementation of e-assessment procedures (see annex B for details)
eBIT LMS administrator	The eBIT LMS administrator is responsible for the configuration and optimal functioning of the LMS platform and for securing the consistency databases in use and physical data storage of information stored in the LMS. He/she will provide expert advise on request to other UCSC staff (and staff of the test centers) on the use of LMS including the do's and don'ts when building e-learning and e-assessment content and how to search the Internet for open (free available) e-learning and e-assessment content He/she will explorer the availability of tools and technologies for e-content and e-assessment as shareware and in the commercial market, the availability of open e-learning and e-assessment content available on the internet. On demand he/she will require additional tools through acquisition or in house development providing additional functionality's needed by the course module owners and content developers. For this purpose the LMS administrator manager will build up and maintain contacts with peer colleagues elsewhere in the global academic society. The eBIT LMS administrator will report to the eBIT project committee
eBIT Instructional Designer	The eBIT Instructional Designer assists the eBIT course coordinator with the development of e-learning and e-assessment content. He/she will directly report to the respective eBIT course coordinator(s).
eBIT system administrator and network manager	The eBIT system administrator and network manager assist the eBIT course administrator and LMS application and data bases manager with making available and securing the continuous operation of needed network services, access to the LMS, authorization of students and the implementation of an adequate back up system securing the physical security of system and application software and data. He/she will report to the eBIT project coordinator

Appendix E: Description of the eBIT e-assessment process

#	Activity cluster	Responsibility of
1	*Request for e-assessment session.* Students shall reply for an e-assessment session by email or by submitting a form available at the eBIT test centers	Student
2	*Validate.* Admission to an e-assessment session will be solely granted to students who are registered on the list of applicants produced by the Learning Management System in use	eBIT e-assessment coordinator
3	*Issue time slot.* Students granted admissions will be provide a time slot for an e-assessment session at one of the eBIT test centers. The time allotted shall, by all reasonable standards, be long enough to allow candidates sufficient time to answer it questions. Students will be informed about the time slot and location of the test center by e-mail.	eBIT e-assessment coordinator
4	*Authentication.* Before the e-assessment session start, the student shall identify him or herself by showing an eBIT or national identification card	eBIT e-assessment coordinator
5	*Authorization.* Authorized students will be provided an password valid during the e-assessment session	eBIT e-assessment coordinator
6	*Registration of attendance.* Students are not allowed to leave the e-assessment session in the first half hour. After this time, permission to leave temporarily will be given only in urgent cases. Attendance of the candidate will be registered including temporarily leave including reason why.	eBIT e-assessment coordinator
7	*Confirm completion.* The student will inform the eBIT course administrator that he or she has completed the e-assessment session. Automatic counter down timer is set by the system and if time out already marked answers are submitted to the system after announcing which confirms the completion	Student
8	*Validate grades. A*fter completion, results of the e-assessment sessions will be evaluated directly with the use of the eBIT e-Assessment system. The results of the automated evaluation will be reviewed by respective responsible UCSC staff.	eBIT course coordinator
9	*Publication of e-assessment results.* Students will be informed by email about the evaluation results of the e-assessment session within …. days after the e-assessment session or results are published on the web/LMS and it is password protected	eBIT course coordinator
10	*Negative evaluation*. In case of an no pass, the student will be entitled to re-apply for re-evaluation session in accordance with article … of the examination regulations,	Student
11	*Positive evaluation*. In case of a pass, a grade will be issue to the student in correspondent to the evaluation results. USCS will register these results accordingly in its administration	eBIT course coordinator

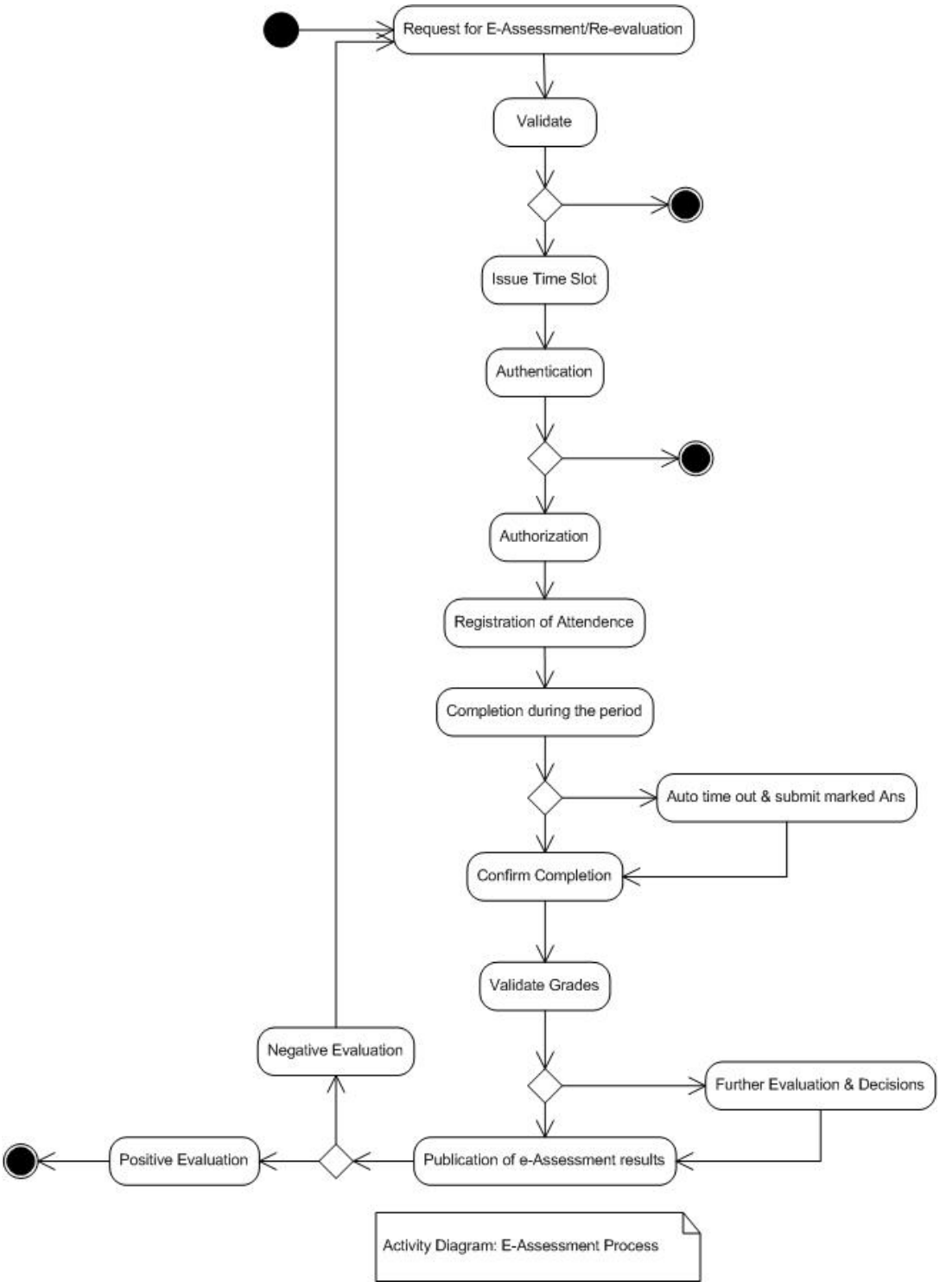


Figure 39: Activity Diagram: E-Assessment Process

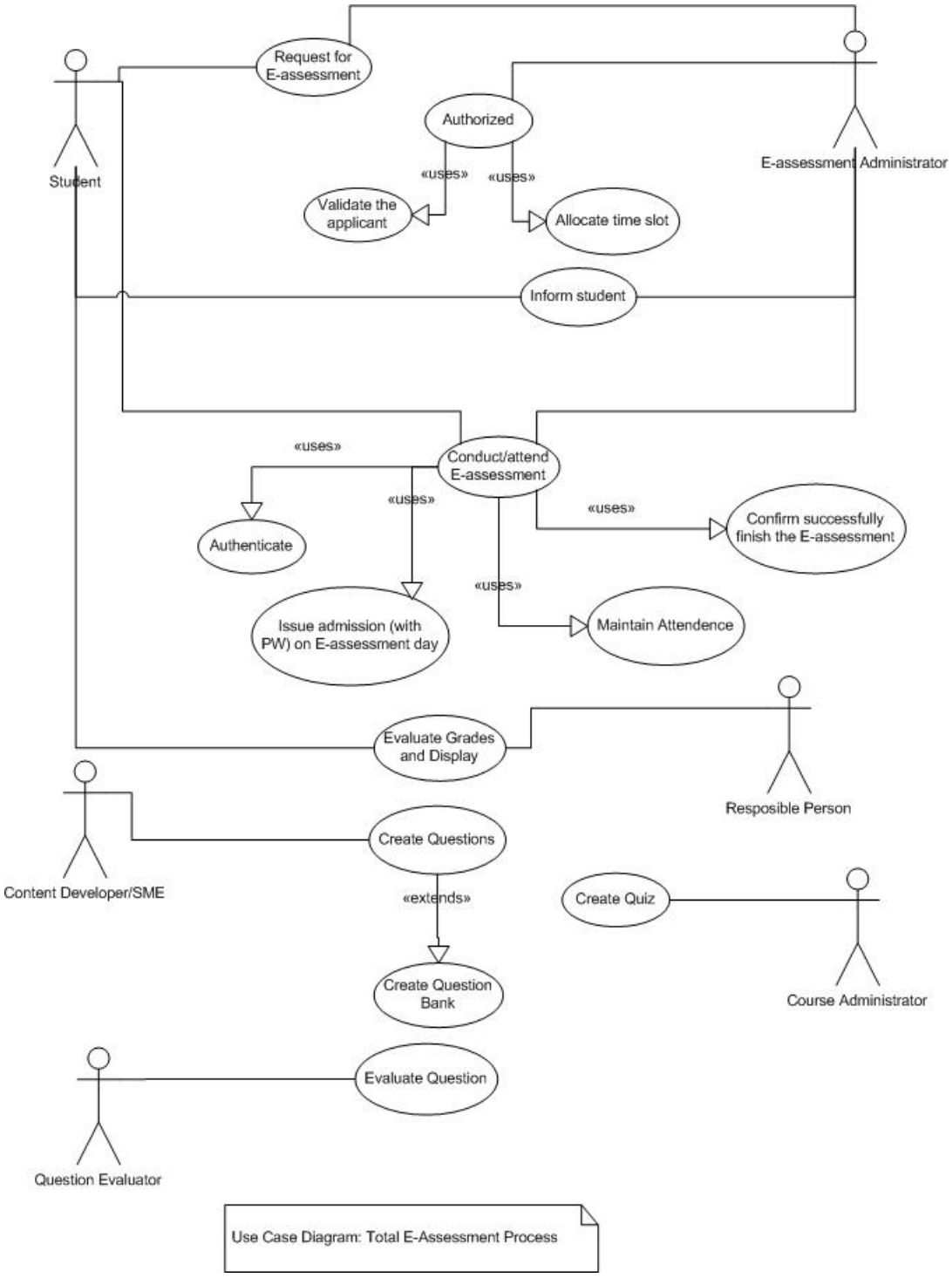


Figure 40: Use case diagram: E-Assessment Process

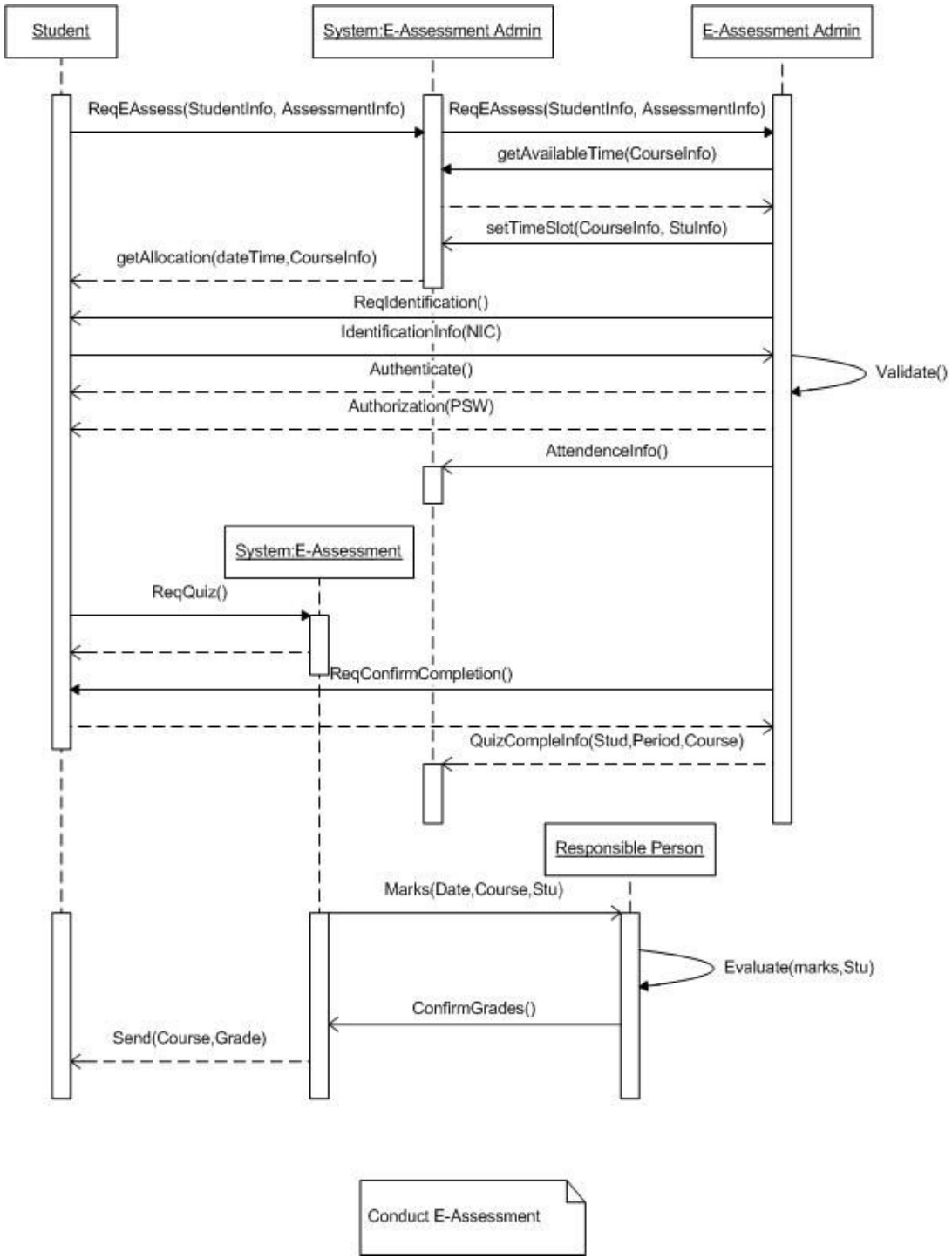


Figure 41: Sequence diagram: E-Assessment Process

Appendix F: Example code

In this appendix some example code has been added. In this example retrieved from the DocumentParser class it is clear how documentation, application of Design By Contract and testing is done. Each class has a class invariant which is tested each time a function is called. As it can be seen in the get_document() function, it is clear what can be expected by calling this function. Also the preconditions are tested. In the parse() function also inline documentation has been added to maximize the readability.

```
/**
 * Class invariant
 */
private function invariant(){
    return $this->xmldocument != null
           && $this->xmlproperties != null
           && ($this->proproot == null ^ $this->isparsed)
           && ($this->docroot == null ^ $this->isparsed)
           && ($this->document == null ^ $this->isparsed);
}

/**
 * Parse the document
 */
public function parse(){
    assert('$this->invariant()');

    //parse files to xmltags
    $this->parse_xmlfiles();

    //parse to documentelement
    $t2eparser = new TagToElementParser($this->docroot,$this->proproot);
    $t2eparser->parse();
    $parseddoc = $t2eparser->get_document();

    //sweep document
    $sweeper = new DocumentSweeper($parseddoc, $this->proproot);
    $sweeper->sweep();
    $sweeper->sweep();
    $this->document = $sweeper->get_document();

    //set parse flag
    $this->isparsed = true;
    assert('$this->invariant()');
}
```

```
/**
 * Get $document variable
 *
 * @return DocumentElement toplevel of document
 */
public function get_document(){
    assert('$this->invariant()');
    assert('$this->isparsed');

    return $this->document;
}
```

Also an example of the testcode has been added. The next few lines of codes are originated from the test_apply() function of the TestDeleteChildPropRule. This is one of the many possible scenarios that are tested in the testing environment.

```
/*
 * Make DocumentElement with 2 childs, parent has no properties
 *
 * Sweeper should remove nothing
 */
$docroot->remove_property('PropertyTest');
$docchild2->add_property($prop3);

$sweeper = new DocumentSweeper($docroot,$this->props);
$sweeper->sweep();
$document = $sweeper->get_document();

$this->assertEqual(count($document->get_properties()),0);
$childs = $document->get_next_level();
$this->assertEqual(count($childs[0]->get_properties()),1);
$this->assertEqual(count($childs[1]->get_properties()),1);
```

9. Model Solution document structure - References

Document element	Property	Description
Autotext		Text that is automatically generated by word. For example the page numbers
	Autotext	String value. The automatically generated text
Body		The root of a word document
	No properties	
Border group		Represents a hint for transforms describing the border, shading, and positioning of groups of paragraphs with similar properties
	Border color	Hexadecimal RGB color value
	Border shadow	Indicates the shadow type of the border
	Border style	Indicates the line type of the border
	Border width	Indicates the width of the border
Borders		Represents describes the borders for the group
	No properties	
Bottom border		Represents the bottom border
	No properties	
Column		Represents a table column
	No properties	
Footer		Represents the footers that appear at the bottom of the pages
	No properties	
Header		Represents the headers that appear at the top of the pages
	No properties	
Left border		Represents the left border
	No properties	
List		Represents a list
	List style	The style of the way a list element is presented *bulleted* Every list element starts with a bullet *numbered* Every list element starts with a number
Paragraph	**Full text**	Represents the text content of the paragraph

	Justification	Represents the justification of the absolute-positioning frame
	Bold	Specifies whether the text is made bold
	Italic	Specifies whether the text is made italic
	Underline	Specifies whether the text is underlined
	Page break before	Represents Page Break Before option: Forces a page break before this paragraph
	Font	The font of the text
	Font size	The size of the font used
	Color	A hexadecimal RGB value specifying the text color
	Ident left	Integer value representing paragraph indentation. Specifies space between left margin and text. Negative values move text into margin
	Ident right	Integer value representing paragraph indentation. Specifies space between right margin and text. Negative values move text into margin
	Ident first line	Integer value. Specifies indent for first line only
	Spacing before	Represents spacing between lines and paragraphs.Gets or sets amount of space above paragraph
	Spacing after	Represents spacing between lines and paragraphs.Gets or sets amount of space below paragraph
	Spacing between	Represents spacing between lines and paragraphs.Gets or sets amount of space between paragraphs
Picture		Represents a picture or other binary object that appears at this point in the document
	No properties	
Right border		Represents the right border
	No properties	
Row		Represents the table row data
	No properties	
Tab		Contains elements that specify tab stop position and alignment for a single tab stop
	Tab type	
		Left
		Center
		Right
	Tab position	Specifies the position of a tab stop.
Table		Defines the table to contain cells.
	No properties	
Tabs		Contains a collection of Tab elements.
	No properties	
Text		Represents text inside a paragraph.
	Text	Specifies the content of the text
	Bold	Specifies whether the text is made bold
	Italic	Specifies whether the text is made italic
	Underline	Specifies whether the text is made underline
	Font	The font of the text

	Font size	The size of the font used
	Color	A hexadecimal RGB value specifying the text color
	Tab before text	Boolean value.Specifies whether there is a tab before the text
	Hyphenation	Specifies the hyphenation style *None* *Normal* *add-before* *change-before* *change-after* *delete and change*
Top border		Represents the top border
	No properties	