A study of change management of Uml diagrams

Ming Chang

University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/2221

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
A STUDY OF CHANGE MANAGEMENT OF UML DIAGRAMS

by

Ming Chang

Bachelor of Science, Computer Science
University of Nevada, Las Vegas
2005

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2007
UMI Number: 1452232

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI

UMI Microform 1452232
Copyright 2008 by ProQuest LLC.
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Thesis Approval
The Graduate College
University of Nevada, Las Vegas

SEPTEMBER 25TH, 2007

The Thesis prepared by
MING CHANG

Entitled
A STUDY OF CHANGE MANAGEMENT OF UML DIAGRAMS

is approved in partial fulfillment of the requirements for the degree of
MASTEr OF SCIENCe IN COMPUTER SCIENCe

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
ABSTRACT

A Study of Change Management of UML Diagrams

by

Ming Chang

Dr. Thomas A. Nartker, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

One of the most critical factors controlling the success of software product development efforts is the management of product changes during development. It is characteristic of software development that everything changes, or can change, continually throughout the process. Requirements can change, the architecture can change, designs can change, and for each of these, the implementation must change. Of course, some changes are natural and necessary. However, it is very common for teams to propose excessive changes that easily overwhelm the budget allocated for the product. Development efforts fail because the effect of each of the changes proposed is not understood until well after it is too late to salvage the project.

Because of this, "change control" is now recognized as a needed component of most software development efforts. It is standard practice to employ a "configuration management system" that keeps a textual database containing a description of successive versions of the product. If the important changes are textual, they can easily be observed by comparing the text recorded by the configuration management system. Using the most
modern development tools however, product changes are represented as changed diagrams.

In this thesis we review the literature of diagram differencing tools and propose a new approach to presenting the differences between two successive "versions" of UML diagrams. We conclude with a discussion of the strengths and weaknesses of our approach and contrast our results with previous attempts at solving this problem.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2 RELATED WORK</td>
<td>3</td>
</tr>
<tr>
<td>Ohst’s approach</td>
<td>3</td>
</tr>
<tr>
<td>Xing’s method</td>
<td>5</td>
</tr>
<tr>
<td>Park’s XML-diff</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 3 APPROACHES FOR MANAGING CHANGES</td>
<td>10</td>
</tr>
<tr>
<td>Concurrent Version System</td>
<td>10</td>
</tr>
<tr>
<td>Subversion</td>
<td>11</td>
</tr>
<tr>
<td>Trac System</td>
<td>11</td>
</tr>
<tr>
<td>IBM Rational ClearCase</td>
<td>12</td>
</tr>
<tr>
<td>CHAPTER 4 XMI FORMAT AND ARGOUML</td>
<td>13</td>
</tr>
<tr>
<td>XMI format and XML</td>
<td>13</td>
</tr>
<tr>
<td>ArgoUML</td>
<td>14</td>
</tr>
<tr>
<td>CHAPTER 5 UML DIAGRAMS IN XMI</td>
<td>16</td>
</tr>
<tr>
<td>Class Diagram</td>
<td>16</td>
</tr>
<tr>
<td>Example Class Diagram</td>
<td>17</td>
</tr>
<tr>
<td>Use Case Diagram</td>
<td>17</td>
</tr>
<tr>
<td>Example Use Case Diagram</td>
<td>18</td>
</tr>
<tr>
<td>CHAPTER 6 METHODOLOGY</td>
<td>19</td>
</tr>
<tr>
<td>Determine the state of an UML element</td>
<td>20</td>
</tr>
<tr>
<td>Levels of the UML elements</td>
<td>21</td>
</tr>
<tr>
<td>Differences that are taken into account</td>
<td>22</td>
</tr>
<tr>
<td>Examples of Running the UML Diff</td>
<td>24</td>
</tr>
<tr>
<td>CHAPTER 7 CONCLUSION</td>
<td>30</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>32</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Example of UML diagram created by ArgoUML</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2</td>
<td>UMLDiff from Ohst’s paper</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Xing’s UMLDiff applied in Eclipse</td>
<td>6</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Storing version differences in HIP</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Storing version differences using SVEM</td>
<td>8</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Ticket feature used in Trac System</td>
<td>12</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Two XML file representing the same information</td>
<td>13</td>
</tr>
<tr>
<td>Figure 8</td>
<td>XMI presentation of UML diagram</td>
<td>14</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Elevator class diagram created by ArgoUML version 0.22</td>
<td>17</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Lift Object Use Case diagram created by ArgoUML version 0.22</td>
<td>18</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Parsed XMI file into a TreeView</td>
<td>20</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Hash table that holds the state of each element</td>
<td>21</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Levels in UML elements</td>
<td>22</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Elements in XMI content created</td>
<td>23</td>
</tr>
<tr>
<td>Figure 15</td>
<td>UMLDiff example of Use Case diagram of Robot Mechanism</td>
<td>24</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Example of how “Added” UML element</td>
<td>25</td>
</tr>
<tr>
<td>Figure 17</td>
<td>UMLDiff example of Use Case diagram of Sales Order System</td>
<td>25</td>
</tr>
<tr>
<td>Figure 18</td>
<td>UMLDiff example of Class diagram of Elevator</td>
<td>26</td>
</tr>
<tr>
<td>Figure 19</td>
<td>UML Class diagram of Elevator created using ArgoUML v0.22</td>
<td>26</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Example showing a “Changed” UML element</td>
<td>27</td>
</tr>
<tr>
<td>Figure 21</td>
<td>UMLDiff example of Class diagram of Ordering System</td>
<td>28</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Example of a “Removed” UML element</td>
<td>28</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Example output of UMLDiff with “Added”, “Removed” and “Changed”</td>
<td>29</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to thank Dr. Nartker for being my advisor and his suggestions and comments through out my thesis. Moreover, his teaching on Unified Process helped me understand the importance of Change Control in software projects. I would like to thank Dr. Hsieh for being in my committee, and for giving me enough time and resource to develop my thesis while I was working on another project. I would like to thank Dr. Gewali, Dr. Taghva for being part of my committee.

Lastly, I would like to thank my family for supporting me, and understanding the long hours spent on writing the code and this thesis.
CHAPTER 1

INTRODUCTION

Object-oriented software development, using UML as a notation for visualizing software artifacts\(^1\), has become an international standard for recording progress in developing a software product. Because of this, the important differences between successive versions of a product artifact are diagrammatic and not textual. Thus, the task of controlling changes during development requires the ability to identify the differences between successive versions of UML diagrams.

Of course, the task of identifying changes in a diagram will depend upon the particular diagram representation being used and this representation will be different for different UML tools. In this thesis, we make use of a popular open source software package for drawing UML diagrams called ArgoUML. Because ArgoUML utilizes the XMI standard to represent UML diagrams, it provides a textual representation of each diagram that we propose can be used to identify and characterize diagram differences.

In this thesis, we utilized version 0.22 of ArgoUML to create the diagrams to be compared. Figure 1 shows a UML diagram drawn using the ArgoUML system.

\(^{1}\) The standard product artifacts are a description of product requirements, a description of product architecture, and a description of the design of product components

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
In chapter 2, we review the literature of attempts to compare and illustrate the differences between UML diagrams. In chapter 3, we review different configuration management systems. In chapters 4 and 5, we present the XMI format used by ArgoUML and examples of UML diagrams in XMI format. In chapter 6, we present our methodology for illustrating differences between successive UML diagrams. The source code for this methodology is shown in the Appendix. Finally, in chapter 7, we present a discussion of our method and we compare our method with methods from other authors.
CHAPTER 2

RELATED WORK

Three different authors have addressed the need to identify the differences between versions of UML diagrams.

Ohst’s approach

Ohst addressed the problem of how to detect and visualize differences among versions of UML diagrams [13]. The authors assumed that the software documents, that contain UML information, are stored in XML. The characteristics of the information stored in such a XML file, are separated into two categories; [16] one which has the layout that is semantically relevant, (i.e. Sequence diagrams), and one which has the layout of any diagram other than a Sequence diagram. In addition to the categories above, there are also diagram structures and attribute forms.

The structure of the diagrams consists of the nodes and the relationships linking them. Attributes can be defined as single-valued, multi-valued or reference attributes. Using the same concept, in [12] the authors presented the algorithm for computing the difference of two documents.
Figure 2 shows the approach of Ohst to differentiate the UML, by using lines and colors. The algorithm used by the author is a technique using the XML format of a
Taking advantage of the tree structure allows the authors to easily create an algorithm that will compare the tree structure and its sub-tree inside the UML document.

Xing's method

Xing and Stroulia emphasize the coder's view. The idea they propose is to take two versions of a Java program and reverse engineer them into two class models of the Java system. This is then used as input to the UMLDiff [11]. Because an object-oriented software system is best defined by a Class diagram, the authors only take into account the UMLDiff of two class models.

UMLDiff mentioned in the paper traverses two class models created by the translation of the Java codes, identifies the structure similarity result in a change tree for the two class models. However, there arises a problem when this UMLDiff scheme is used. The structure similarity relies on two thresholds, the rename-threshold and the move-threshold; and both of them are user-defined. The use of the threshold could affect the quality of the result in comparing the two class models. Since each class model might have a different threshold, a new threshold needs to be created for each UMLDiff used.
The benefit of using Xing’s UMLDiff shown in Figure 3 is that not only are structure and similarity managed, but this UMLDiff is elaborated for a coder’s use. Although the similarity is managed, the performance is affected by the choice of thresholds; moreover UMLDiff is limited by a single programming language, Java. This is completely against the principle of UML where UML elements should not be restricted by any programming language.
Park's XML-diff

To manage the changes in XML-based software documents, the authors in [5] demonstrated that the SVEM (Sparsely Version-stamped Edge Model) algorithm is $m$ times better than the HIP [21] algorithm, where $m$ is the average number of children.

A three-tuple attribute is used to handle the changes; the three-tuple contains the version stamp, the edge the version stamp is assigned to, and the type of the version stamp. For each version, the differences are created using the algorithm mentioned in [14]
and stored in a version stamp stack. Because the information in the version stamp stack defines all the changes made between versions, the previous version, or a target version, of current the tree can be easily recreated.

Figure 5 Storing version differences using SVEM

The version stamp stack can be helpful in controlling the changes between XML-based documents. It is worth pointing out that SVEM might work in managing the UML diagrams in XMI but it does not take full advantage of XMI format. This XMI
format for UML diagrams will be discussed in chapter 4.
CHAPTER 3

APPROACHES FOR MANAGING CHANGES

Most of the Software Configuration Management (SCM) tools available record each version of the source code (or documentation) produced. They also keep history information so that developers can go back and review the changes between previous versions. SCM tools have been widely used and are essential for any type of software development. Such tools help developers work together to improve the code and its documentation. Among the most used tools are CVS and Subversion. An emerging tool that is open-source is the Trac System which utilizes a web interface. There is also a popular commercial tool, named Rational ClearCase, which was developed by IBM that provides both management and control of software development.

Concurrent Version System

Concurrent Versions System (CVS) is one of the most used SCM tools. Grune first gave details about the functionality of CVS in [17]. CVS has a repository that will retain all versions of documents. The user can issue a Commit command on a file that will commit the new file to the repository. If a new version has been created then the Update command is needed to merge the new version into the repository.

CVS has been available for free since it was created. It has been the most commonly used SCM tool.
The nature of CVS has some weakness. For instance commits are not atomic which can cause the repository to be left in an intermediate state if the Commit command is interrupted. Also, moving the directory or changing the name is not supported [10].

Subversion

Subversion (SVN) is another SCM tool that is an improved version of CVS; it is designed to be a modern replacement of CVS. SVN is also open source and has most of the features provided by CVS which allows users to easily migrate from CVS to SVN. There are several improvements over CVS; for example commit is atomic therefore an interrupted Commit will not cause the repository to be in an intermediate state. While CVS lacks support on rename and directories, SVN allows metadata, or properties, to be versioned with a file or a directory. The only drawback of SVN compared to CVS is that SVN does not have a commit message for each change made to the repository.

Trac System

The Trac System is a web-based, integrated SCM and project management software tool, which can be used with an SVN repository. Trac helps users track bugs (or issues), making the system easier to use. It also combines the use of Wikipedia with a version control system [18].
Instead of improving the current SVN system, Trac uses SVN to manage the source code and documentation. Versions are stored in the SVN repository, while Trac keeps the bug and issue tracking in a Wikipedia style. The Trac System accesses the repository and retrieves information such as the source code or documentation, the timeline of a project, the changes of source code, and the change history. It then creates an element that is unique to Trac, that is called the "ticket", which is a number assigned to a bug (or issue) as shown in Figure 6.

IBM Rational ClearCase

Rational ClearCase is an SCM tool that is part of the IBM Rational Software family. ClearCase is a commercial product that includes major features of SVN for managing source code and documentation. In addition to the common features in any SCM, ClearCase includes Version Object Base, which is a repository storing versions and metadata associated with these versions. Moreover MultiSite Version Object Base can have multiple replicas at different sites. ClearCase also stores the Configuration Record, that is a trace of the objects or files read throughout the build process. This allows ClearCase to determine whether two derived objects are exactly the same by comparing the Configuration Record [2].
CHAPTER 4

XMI FORMAT AND ARGOUML

Extensible Markup Language Metadata Interchange (XMI) is an Object Management Group standard that makes exchanging metadata information via the Extensible Markup Language possible [6]. XMI is an industry standard and it can be used to represent UML diagrams in a textual format. In this paper ArgoUML version 0.22 will be used to produce XMI files.

XMI format and XML

A standard XML format [19] is used to define XMI format. XMI is a form of XML. XML does not tell how the information is stored or where it should be stored. XMI specifies the exact elements and format that can be used in a particular location of a document.

```
<Auto>
  Kind = "convertible"
  Year = "2002" />
</Car>
```

Figure 7 Two XML file representing the same information

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
In Figure 7 there are two XML files that have the same logic and meaning, but different textual representation. When using XMI to represent a UML diagram, this does not happen. The reason it does not occur is because the XMI standard specifies the exact position that an element can be located at, with defined constraint and format. If the same UML elements are present then the elements will have the same boundary as shown in Figure 8.

```xml
<?xml version='1.0' encoding='UTF-8'?><XMI xmli.version='1.2' xmlns=UML='org.omg.xmi.namespace.UML' timestamp='Wed Apr 18 12:34:26 PDT 2007'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>ArgoUML (using Netbeans XMI Writer version 1.0)</XMI.exporter>
      <XMI.exporterVersion>0.2G.x</XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UHL" xmi.version="1.4"/>
  </XMI.header>
  <XMI.content>
    <UML:Model xmi.id='-125--40-23-70-32784a:112062d1085:-8000:000000000000000077B:' name='untitledModel' isSpecification='false' isRoot='false' isLeaf='false' isAbstract='false'>
      <UHL:Namespace.ownedElement>
        <UML:Actor xmi.id='-125--40-23-70-32784a:112062d1085:-8000:000000000000000077C:' name='User' isSpecification='false' isRoot='false' isLeaf='false' isAbstract='false'>
          <UML:ModelElement.clientDependency>
            <UML:Dependency xmi.idref='-125--40-23-70-32784a:112062d1085:-8000:0000000000000000781'/>
          </UML:ModelElement.clientDependency>
        </UML:Actor>
      </UML:ModelElement.clientDependency>
    </UML:Model>
  </XMI.content>
</XMI>
```

Figure 8 XMI presentation of UML diagram

ArgoUML

ArgoUML is an open source UML modeling tool written in Java that supports standard UML1.4 diagrams [20]. ArgoUML version 0.22 requires Java 1.4 or Java 5 in order to execute. It supports modeling of Class diagrams, State Chart diagrams, Activity diagrams, Use Case diagrams, Collaboration diagrams, Deployment diagrams and
Sequence diagrams. This paper will only consider the Class and the Use Case diagrams created by ArgoUML. We will take advantage of ArgoUML’s ability of importing and exporting UML 1.4 formats using XMI 1.2.

ArgoUML is a major open-source UML modeling tool which has several plug-ins and subprojects being developed. This includes natural language support (i.e. Portuguese and French) as well as programming languages such as C++, C# and Ruby. The programming language support allows users to generate source files from ArgoUML. For some programming subprojects, a UML model and diagram can be generated from the source code.
CHAPTER 5

UML DIAGRAMS IN XMI

UML diagrams created by ArgoUML version 0.22 use UML version 1.4 and format XMI version 1.2. Diagrams imported to ArgoUML can utilize either XMI 1.1 or XMI 1.2. To avoid version problems in this paper, only XMI 1.2 is used in all UML elements created.

Class Diagram

The XMI representation of each UML element of a Class diagram is in fixed format. There are several elements that belong to the Class diagram family. The following are the major elements:

- UML:Class
- UML:DataType
- UML:Package
- UML:Operation
- UML:Attribute
- UML:Extend
- UML:Include
Example Class Diagram

Figure 9 Elevator class diagram created by ArgoUML version 0.22

Use Case Diagrams

The XMI representation of each UML element in a Use Case diagram is also fixed in format. The following major elements belong to the Use Case family:

- UML:UseCase
- UML:Actor
- UML:Association
- UML:Dependency
- UML:Generalization
Example Use Case Diagram

Figure 10 Lift Object Use Case diagram created by ArgoUML version 0.22
CHAPTER 6

METHODOLOGY

In this paper, an algorithm is created to retrieve the differences between two versions of a UML diagram using the XMI 1.2 format. The two XMI files are created by ArgoUML v0.22 and the algorithm is implemented using C#. The first step of this algorithm is parsing both of the XMI files into a TreeView object in C#; this can be done easily by using any XML parsing algorithm available. The result of parsing an XMI file is shown in Figure 11.
Determine the state of an UML element

In order to facilitate the comparison of two XMI documents, before the comparing algorithm is initiated, we need to determine the state of each UML element. There consist of three states for each UML element, and they are Removed UML element, Added UML element and Changed UML element.

First we create one hash table similar to what is shown in Figure 12, for each UML element that is part of a Class or Use Case diagram, and then we run through a loop for the first XMI document. For each UML element present, it is stored into a hash table, using a unique identification number (ID) as the key and the number “1” as the value that such key points to. Next we run through the same loop for the second version of the XMI.
document. Before adding the information into the hash table we compare whether the ID of the current UML element is present in the hash table. If the ID is not present then this particular UML element is added to the hash table with ID as the key and the number “2” as the value, meaning the element is only present in the second version. On the other hand, if the ID is present in the hash table, then in the hash table the element is replaced by ID as key and number “0” as the value. This implies this UML element is present in both the first and the second version of the XMI document.

125--40-23-70-32784a112062d1085--8000:0000000000000780' -> "0"
125--40-23-70-32784a112062d1085--8000:0000000000000787' -> "1"
125--40-23-70-32784a112062d1085--8000:000000000000077c' -> "2"

In both versions       Only in version 1   Only in version 2

Figure 12 Hash table that holds the state of each element

Levels of the UML elements

Even though there can be multiple levels within the XMI representation, only three of those levels shown in Figure 13 are important in this algorithm. In these three levels, the first contains the type of UML element, and the second level contains the ID and several other attributes. The details of each UML element are explained in chapter 4. Finally, the last level has the value for each attribute in level 2.
Differences that are taken into account

Inside the XMI document all the relevant UML elements are under “UML:Namespace.ownedElement”. Thus, the algorithm only loops through and compares the difference between the elements under “UML:Namespace.ownedElement”. As shown in Figure 14, the elements that are not relevant are not compared because they are identical in both versions of the XMI file.
Using the method described earlier, we determined the state of each UML element. If an ID in the hash table returns “1”, this means that the UML element is only present in the first version of the XMI document. Alternatively, if this ID has returned a value of “2” then the UML element is only available in second version of the XMI document. Thus, when “1” is returned, we can mark this UML element as “Removed”. When “2” is returned, “Added” will be marked. The other important part of the comparison is when the ID has returned a value of “0”. This means that the UML element is in both the first and second version. If UML element can be found in both versions, this requires comparing all children within the UML element, which uses a function described below.

The function that compares children in two XMI documents has a loop that compares Level 1 and Level 2 of UML elements. On the first run and for each element in Level 2 a recursive call is made to the function that will compare Level 2 and Level 3. When the ID
of an element from Level 1 of the first version matches with the ID of an element from Level 1 of the second version, then Level 2 and Level 3 is compared side by side.

A color is associated with all three types of differences. They are: Added, Removed and Changed. When a UML element is Added or Removed, the element will be highlighted by the color blue or green, respectively. Changed not only highlights the Level 1 but also Level 2 and Level 3 of the UML element. If a Level 3 is Changed then it will have “changed from:” value of first version “to:” the value of second version.

Examples of Running the UML Diff

Figure 15, Figure 17, Figure 18 and Figure 21 illustrate how UMLDiff marks the presence of “Added”, “Removed” or “Changed” elements. All are highlighted in a different color. Blue for “Added”, green for “Removed” and red for “Changed”.

Figure 15 UMLDiff example of Use Case diagram of Robot Mechanism
BLUE - Added

Figure 16 Example of how “Added” UML element

Figure 17 UMLDiff example of Use Case diagram of Sales Order System

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 18 UMLDiff example of Class diagram of Elevator

Figure 19 UML Class diagram of Elevator created using ArgoUML v0.22
Figure 19, shows how a UML diagram is presented by ArgoUML. It is represented internally as a standard XMI file which is the input to UMLDiff. An example output is shown in Figure 20.

Figure 20 Example showing a “Changed” UML element
Figure 21 UMLDiff example of Class diagram of Ordering System

Figure 22 Example of a “Removed” UML element
Figure 16, Figure 20, and Figure 22 shows “Added”, “Changed” and “Removed”, respectively. Figure 23 shows a case where all three types of states are present in the same UMLDiff.

Figure 23 Example output of UMLDiff with “Added”, “Removed” and “Changed”
CHAPTER 7

CONCLUSION

There are three different UMLDiff methods produced by other authors described in this paper. First, by coloring UML components and using a different line representation, Ohst provides the advantage of a pictorial presentation of diagram changes. However the format used to keep UML information is not published, which makes it difficult for any other use of the method proposed. In addition, Ohst’s algorithms were not described and are not available for study and modification.

Xing’s UMLDiff focuses on a coder’s view, and the algorithm relies heavily on a rename threshold and a move threshold. Both the rename and move thresholds introduce some inefficiency because the thresholds must be reevaluated for each UMLDiff. In addition, this method is strictly for the Java programming language.

Park’s method has the advantage of having a space-efficient algorithm. He introduces a simple and space efficient “version stamp stack”, that keeps track of the changes between versions. The “version stamp stack” will store the change information in the stack, which can be used to regenerate the file compared. Because Park uses XML, this method also requires ArgoUML or similar UML diagram drawing package.

In this thesis, we propose a method to compare the differences of two UML diagrams in XMI format, focusing on Class and Use Case diagrams. Even though only Class and Use Case are considered, adding support for other UML diagrams is easy. For Sequence

30
diagrams, Activity diagrams, State Chart diagrams or other UML diagrams, a dedicated
hash table for each type of diagram is needed. The hash table needed can be determined
by examining the XMI format of a diagram.

The only restriction of our method is that XMI files corresponding to all
diagrams must be available. We choose ArgoUML as our UML diagram drawing tool
because it generates XMI files. This restriction is minimized because ArgoUML is an
Open Source system.
BIBLIOGRAPHY

11. Zhenchang Xing and Eleni Stroulia “UMLDiff: An Algorithm for Object-Oriented Design Differencing” ASE’05, November 7-11, 2005
14. Grégory Cobena, Serge Abiteboul and Amélie Marian “Detecting Changes in XML Documents” Proceedings of the 18th International Conference on Data Engineering (ICDE’02)
APPENDIX I

DETECT THE PRESENCE OF A UML ELEMENT

// Decide whether to put "V1-exist"(1) or "V2-exist"(2) or both(0) in their proper
// HashTable
private void PutV1V2(bool isV1, string hashName, string id)
{
    // isV1 is true when we have to put "V1"
    // hashName is the hash we put to
    // Switch + UML:Model ( id <- "V1_ID or "V2_ID")
    // if isV1 == true
    // then HashTable.Add(id,1)
    // else
    // if HashTable.ContainsKey(id) == false
    // then HashTable.Add(id,2)
    // else
    // Then HashTable.Remove(id)
    // HashTable.Add(id,0)

    try
    {
        switch (hashName)
        {
            case "UML:UseCase":
                if (true == isV1)
                    UseCase.Add(id, 1);
                else
                {
                    if (UseCase.ContainsKey(id))
                    {
                        UseCase.Remove(id);
                        UseCase.Add(id, 0);
                    }
                    else
                        UseCase.Add(id, 2);
                }
                break;
            case "UML:Actor":
            ...
if (true == isV1)
    Actor.Add(id, 1);
else
{
    if (Actor.ContainsKey(id))
    {
        Actor.Remove(id);
        Actor.Add(id, 0);
    }
    else
        Actor.Add(id, 2);
}
break;
case "UML:Association":
if (true == isV1)
    Association.Add(id, 1);
else
{
    if (Association.ContainsKey(id))
    {
        Association.Remove(id);
        Association.Add(id, 0);
    }
    else
        Association.Add(id, 2);
}
break;
case "UML:Dependency":
if (true == isV1)
    Dependency.Add(id, 1);
else
{
    if (Dependency.ContainsKey(id))
    {
        Dependency.Remove(id);
        Dependency.Add(id, 0);
    }
    else
        Dependency.Add(id, 2);
}
break;
case "UML:Generalization":
if (true == isV1)
    Generalization.Add(id, 1);
else
{
if (Generalization.ContainsKey(id))
{
    Generalization.Remove(id);
    Generalization.Add(id, 0);
}
else
    Generalization.Add(id, 2);
break;
case "UML:Extend":
    if (true == isV1)
        Extend.Add(id, 1);
    else
    {
        if (Extend.ContainsKey(id))
        {
            Extend.Remove(id);
            Extend.Add(id, 0);
        }
        else
            Extend.Add(id, 2);
    }
    break;
case "UML:Include":
    if (true == isV1)
        Include.Add(id, 1);
    else
    {
        if (Include.ContainsKey(id))
        {
            Include.Remove(id);
            Include.Add(id, 0);
        }
        else
            Include.Add(id, 2);
    }
    break;
case "UML:Class":
    if (true == isV1)
        Class.Add(id, 1);
    else
    {
        if (Class.ContainsKey(id))
        {
            Class.Remove(id);
            Class.Add(id, 0);
        }
    }
36
else
Class.Add(id, 2);
}
break;
case "UML:DataType":
if (true == isV1)
DataType.Add(id, 1);
else
{
if (DataType.ContainsKey(id))
{
DataType.Remove(id);
DataType.Add(id, 0);
}
else
UseCase.Add(id, 2);
}
break;
case "UML:Package":
if (true == isV1)
Package.Add(id, 1);
else
{
if (Package.ContainsKey(id))
{
Package.Remove(id);
Package.Add(id, 0);
}
else
Package.Add(id, 2);
}
break;
case "UML:Model":
if (true == isV1)
{
MainHashTable.Add("V1_ID", id);
}
else
{
MainHashTable.Add("V2_ID", id);
}
break;
catch (NullReferenceException nre)
{
    MessageBox.Show("Duplicate Key");
    MessageBox.Show(nre.Message);
}
catch (ArgumentException ae)
{
    MessageBox.Show("Key == NULL, Already in Table");
    MessageBox.Show(ae.Message);
}
private void compareSubTree(TreeNodeCollection treeC1, TreeNodeCollection treeC2, TreeNode tempNodeLevel_1)
{
    TreeNode tempNodeLevel_2 = new TreeNode();
    TreeNode tempNodeLevel_3 = new TreeNode();
    TreeNode tempNodeLevel_4 = new TreeNode();
    TreeNode tempNodeLevel_5 = new TreeNode();
    TreeNode recursiveNodeLevel_1 = new TreeNode();
    TreeNode recursiveNodeLevel_2 = new TreeNode();
    Hashtable recursiveNodeHash1 = new Hashtable();
    Hashtable recursiveNodeHash2 = new Hashtable();
    Hashtable recursiveNodeHash1_temp = new Hashtable();
    Hashtable recursiveNodeHash2_temp = new Hashtable();
    Hashtable outHashTable = new Hashtable(); // Not Used
    Hashtable inHashTableV1 = new Hashtable();
    Hashtable inHashTableV2 = new Hashtable();
    UML_OWNER_SPACE = false;
    bool UML = false;
    bool UML_main = false;
    bool xmi_idref = false;
    bool inRun = false;
    int i;
    string[] split1 = null;
    string[] split2 = null;
    string str1;
    string str2;
    char delimstr = ':';
    // Level 2, Level 3
    i = 0;
    foreach (TreeNode node_1_out in treeC1)
    {
        if (node_1_out.Text != "UML:Namespace.ownedElement")
        {
            split1 = node_1_out.Text.Split(delimstr);
            if (split1[0] == "UML")
            {
...
{  
    UML = true;
    if (i == 0)
    {
        UML_main = true;
    }
    if (node1_out.Nodes[0].Text.ToString() == "xmi.idref")
    {
        xmi_idref = true;
        if (true == firstRun)
        {
            recursiveNodeHash1.Add(node1_out.Nodes[0].Nodes[0].Nodes[0].Text.ToString(), node1_out);
        }
    }
    recursiveNodeHash1.Add(node1_out.Nodes[0].Nodes[0].Nodes[0].Text.ToString(), node1_out);
    
    else
    {
        recursiveNodeHash1.Add(node1_out, node1_out.Nodes[0].Nodes[0].Nodes[0].Text.ToString());
    }
    else
    {
        if (true == firstRun)
        {
            recursiveNodeHash1.Add(node1_out.Nodes[0].Nodes[0].Text.ToString(), node1_out);
        }
    }
    else
    {
        UML = false;
        recursiveNodeHash1.Add(node1_out.Text.ToString(), node1_out.Nodes[0].Text.ToString());
    }
    
    else
    {
        UML_OWN_SPACE = true;
    }
}
i++;

//Level 2, Level 3
i = 0;
foreach (TreeNode node2_out in treeC2)
{
    if (node2_out.Text != "UML:Namespace.ownedElement")
    {
        split2 = node2_out.Text.Split(delimstr);
        if (split2[0] == "UML")
        {
            UML = true;
            if (i == 0)
            {
                UML_main = true;
            }
        }
        if (node2_out.Nodes[0].Text.ToString() == "xmi.idref")
        {
            xmi_idref = true;
            if (true == firstRun)
            {
                recursiveNodeHash2.Add(node2_out.Nodes[0].Nodes[0].Text.ToString(), node2_out);
            }
            else
            {
                recursiveNodeHash2.Add(node2_out, node2_out.Nodes[0].Nodes[0].Nodes[0].Text.ToString());
            }
        }
    }
    else
    {
        if (true == firstRun)
        {
            recursiveNodeHash2.Add(node2_out.Nodes[0].Nodes[0].Text.ToString(), node2_out);
        }
        else
        {
            recursiveNodeHash2.Add(node2_out, node2_out.Nodes[0].Nodes[0].Text.ToString());
        }
    }
}
else
{
    UML = false;
}
recursiveNodeHash2.Add(node2_out.Text.ToString(),
node2_out.Nodes[0].Text.ToString());
}
else
{
    UML_OWNER_SPACE = true;
}
i++;
}
//Compare V1 V2
foreach (DictionaryEntry de1 in recursiveNodeHash1)
{
    foreach (DictionaryEntry de2 in recursiveNodeHash2)
    {
        if (true == UML)
        {
            if (de1.Key.ToString() == de2.Key.ToString()) //if "id" == "id"
            {
                str1 = de1.Value.ToString();
                if (firstRun == true)
                {
                    tempNodeLevel_2 = new TreeNode();
                    tempNodeLevel_2.Text = str1;
                    tempNodeLevel_1.Nodes.Add(tempNodeLevel_2);
                }
            }
            else
            {
                tempNodeLevel_2 = new TreeNode();
                tempNodeLevel_2.Text = de1.Key.ToString();
                tempNodeLevel_2.ForeColor = Color.Black;
                tempNodeLevel_3.ForeColor = Color.Black;
            }
        }
        else
        {
            tempNodeLevel_2 = new TreeNode();
            tempNodeLevel_3 = new TreeNode();
            if (de1.Value.ToString() == de2.Value.ToString())
            {
                tempNodeLevel_3.Text = de1.Value.ToString();
                tempNodeLevel_2.Text = de1.Key.ToString();
                tempNodeLevel_2.ForeColor = Color.Black;
                tempNodeLevel_3.ForeColor = Color.Black;
                tempNodeLevel_1.ForeColor = Color.Black;
            }
        }
    }
}
}
tempNodeLevel_3.ForeColor = Color.Red;
tempNodeLevel_2.Text = de1.Key.ToString();
tempNodeLevel_2.ForeColor = Color.Black;
tempNodeLevel_1.ForeColor = Color.Red;
}
tempNodeLevel_2.Nodes.Add(tempNodeLevel_3);
tempNodeLevel_1.Nodes.Add(tempNodeLevel_2);
}
else
{
    if (true == firstRun)
    {
        if (recursiveNodeHash1.ContainsKey(de2.Key.ToString()) == false) // problem -> always goes in
        {
            if (recursiveNodeHash1_temp.ContainsKey(de2.Key) == false)
            {
                try
                {
                    recursiveNodeHash1_temp.Add(de2.Key, de2.Value);
                    if (false == xmi_idref && true == firstRun)
                    {
                        tempNodeLevel_2 = (TreeNode)((TreeNode)de2.Value).Clone();
                        if (tempNodeLevel_2.Nodes[1].Text.ToString() == "name")
                        {
                        }
                    }
                    tempNodeLevel_2.ForeColor = Color.Blue;
                    tempNodeLevel_1.Nodes.Add(tempNodeLevel_2);
                }
                catch (ArgumentException ae)
                {
                    MessageBox.Show("CompareSubTree -for-for-" + ae.Message);
                }
            }
        }
    }
}
else if (false == UML && false == firstRun)
{

}

43
if (del.Key.ToString() == de2.Key.ToString())
{
    tempNodeLevel_2 = new TreeNode();
    tempNodeLevel_3 = new TreeNode();
    if (del.Value.ToString() == de2.Value.ToString())
    {
        tempNodeLevel_2.Text = del.Key.ToString();
        tempNodeLevel_3.Text = del.Value.ToString();
        tempNodeLevel_2.ForeColor = Color.Black;
        tempNodeLevel_3.ForeColor = Color.Black;
        tempNodeLevel_2.Nodes.Add(tempNodeLevel_3);
        tempNodeLevel_1.Nodes.Add(tempNodeLevel_2);
    }
    else //Changed
    {
        tempNodeLevel_2.Text = del.Key.ToString();
        tempNodeLevel_2.ForeColor = Color.Red;
        tempNodeLevel_3.ForeColor = Color.Red;
        tempNodeLevel_1.ForeColor = Color.Red;
        tempNodeLevel_2.Nodes.Add(tempNodeLevel_3);
        tempNodeLevel_1.Nodes.Add(tempNodeLevel_2);
        // Use color RED to define CHANGED
    }
}
}

if (true == UML)
{
    if (true == firstRun)
    {
        if (!recursiveNodeHash2.ContainsKey(de1.Key) && !recursiveNodeHash2_temp.ContainsKey(de1.Key))
        {
            try
            {
                recursiveNodeHash2_temp.Add(de1.Key, de1.Value);
                if (false == xmi_idref && true == firstRun)
                {
                    tempNodeLevel_2 = (TreeNode)((TreeNode)de1.Value).Clone();
                }
            }
        }
    }
}
if (tempNodeLevel_2.Nodes[1].Text.ToString() == "name")
{
    tempNodeLevel_2.Text = tempNodeLevel_2.Text.ToString() + " / " +
    tempNodeLevel_2.Nodes[1].Nodes[0].Text.ToString();
}

tempNodeLevel_2.ForeColor = Color.Green;
tempNodeLevel_1.Nodes.Add(tempNodeLevel_2);
}
catch (ArgumentException ae)
{
    MessageBox.Show("CompareSubTree -for-" + ae.Message);
}
}
VITA

Graduate College
University of Nevada, Las Vegas

Ming Chang

Local Address:
1333 Del Mar St.
Apt. 4
Las Vegas, NV 89119

Home Address:
R. Rui Barbosa 1061
Apt. 603, Centro
Foz do Iguassu, PR, Brasil 85851-170

Degrees:
Bachelor of Science, Computer Science, 2005
University of Nevada, Las Vegas

Thesis Title: A Study of Change Management of UML Diagrams

Thesis Examination Committee:
Committee Chair, Dr. Thomas Nartker, Ph.D.
Committee Member, Dr. Lexmi P. Gewali, Ph.D.
Committee Member, Dr. Kazem Taghva, Ph.D.
Graduate Faculty Representative, Dr. Hsuan-Tsung (Sean) Hsieh, Ph.D.