Generating Class Models Using Binary Space Partition Algorithm

Kashif Hameed and Imran Sarwar Bajwa

Abstract. In this paper, we address a challenging task of automat generation of UML class models. In conventional CASE tools, the export facility does not export the graphical information that explains the way UML class elements (such as classes, associations, etc) are represented and laid out in diagrams. We address them problem by presenting a novel approach for automatic generation of UML class diagrams using the Binary Space Partitioning (BSP) tree data structure. A BSP tree captures the spatial layout and spatial relations in objects in a UML class model drawn on a 2-D plane. Once the information of a UML model is captured in a BSP tree, the same diagram can be re-generated by efficient partitioning of space (i.e. regions) without any collision. After drawing UML classes, the associations, aggregations and generalisations are also drawn between the classes. The presented approach is also implemented in VB.NET as a proof of concept. The contribution does not only assist in diagram interchange but also improved software modeling.

Keywords: UML Class Models, Binary Space Partition Tree, XMI, XML.

1 Introduction

Since the emergence of the Object Oriented Modeling (OOM), the software design patterns have been improved to assist the programmers to address the complexity

Kashif Hameed
Department of Computer Science, The Islamia University of Bahawalpur,
63100, Bahawalpur, Pakistan
e-mail: gentle_kashif@yahoo.com

Imran Sarwar Bajwa
School of Computer Science, University of Birmingham, B15 2TT, Birmingham, UK
e-mail: i.s.bajwa@cs.bham.ac.uk

R. Lee (Ed.): Computer and Information Science 2012, SCI 429, pp. 1–13.
springerlink.com

of a problem domain in a better way. The OOM suggests handling a problem as a set of related and interacting Objects instead of considering as a set of functions that can be performed. A key feature of OOM is re-usability of same piece of information. In OOM, Unified Modeling Language (UML) based graphical notation is used to represent a model or a schema. There are various CASE tools such as Rational Rose, USE, Enterprise Architect, ArgoUML, Altova, Smart Draw, MS Visio, etc. All these tools provide a facility to export metadata of a UML class model using XML Metadata Interchange (XMI) [1] representation. XMI provides a standardized representation of the metadata information in a UML model so that it may be exchanged across many industries and operating environments, different UML tools and other tools that are capable of using XMI. XMI uses XML (Extensible Markup Language) representation to transport information that is highly internally referential.

Considering the undisputable significance of XMI, most of the CASE tools facilitate a user to import and export UML class models in XMI format. However, there is a limitation in current implementation of import/export facility provided by all the CASE tools that only metadata (names of classes, attributes, methods, associations, etc) of a UML class model can be exported or imported. Due to this limitation, a complete UML class diagram can not be interchanged among various CASE tools and it means that UML class diagram dawn in a CASE tool cannot be reused in other CASE tools. This limitation nullifies the basic principal of OOM that is reusability of information. A principal reason of this limitation is that the graphical visualization of a model from XMI is not possible since XMI has no graphic information [19] such as where to draw a model element and how to avoid overlapping of the model elements. In absence of this facility, the real power of XMI remains un-explored.

To address this challenging task, we present an approach for automatic generation of UML class diagrams from XMI using the Binary Space Partitioning (BSP) tree [20]. The BSP trees are typically used in the field of computer graphics to capture graphical information of a 2-D diagram. A BSP tree captures the spatial layout and spatial relations in objects in a UML class model drawn on a 2-D plane. Once the information of a UML model is captured in a BSP tree, the same diagram can be re-generated by efficient partitioning of space (i.e. regions) without any collision. After drawing UML classes, the associations, aggregations and generalisations are also drawn between the classes. We have also implemented the presented approach in VB.NET as a proof of concept and we have also solved a case study to validate the performance of our approach.

Rest of the paper is ordered into various sections: Section 2 highlights the work related to the presented research. Section 3 explains algorithm used for space portioning and its implementation is explained in Section 3. Section 4 presents a case study that manifests the potential of the presented approach in real time software engineering. The paper is concluded with the possible future enhancements.

2 Related Work

Due to limitation of XMI, diagram interchange is not possible in real meanings. Not very much work has been presented to address the limitation of XMI. To address the same issue, Marko [19] proposed some major changes in XMI metamodel to store graphical information of a UML model. However, we think that this is not a good solution due to the reason that the XMI metamodel will become far complex with addition of graphical information and the handling of XMI will become more difficult. The gap in presented work to-date was the real motivation for the presented work.

Similarly, to generate UML diagrams from XML data, Mikael, et al [9] presented an algorithm in 2001. The presented work was primarily focusing on the use of the generated diagrams for the conceptual design of (virtual) a data warehouses with respect to the web data. Another major intention of the research was to support the On-Line Analytical Processing (OLAP) based on web data. However, according to best of our knowledge the generation of UML class diagrams from XMI representation is a novel idea and no approach/tool has been presented to date for this transformation.

Some work has also been presented in automated UML class diagram generation from a natural language (NL) available in [10], [13-18], [26-28]. These approaches extract UML class elements from NL specification of software requirements and generate UML class diagrams. Such work was also the motivation of the presented approach.

To our best of knowledge, no work has been presented yet for generation of UML graphical models from the XMI presentation. Missing support for XMI to UML models transformation breaches a gap in XMI and UML. To fill this gap, there is need of an approach for automated transformation of XMI to UML to make XMI more effective and powerful.

3 Generating UML Class Diagrams from BSP Tree

The presented approach works in two steps. In first step, a BSP tree is generated from XMI representation of a UML class model. In second step, BSP tree is traversed and the UML diagram is generated. The presented approach can process XMI 2.1 format. We have chosen XMI 2.1 format as all major tools such as ArgoUML, Enterprise Architect, USE, Altova UModel support XMI 2.1 format. User provides the input in the form of XMI (.xml) file using the interface provided in the typical software system. Fig. 1 represents the overview of all steps involved in XMI to UML class diagram transformation. Following are details of the involved steps in generation of UML class diagrams from XMI 2.1 using BSP tree data structure.

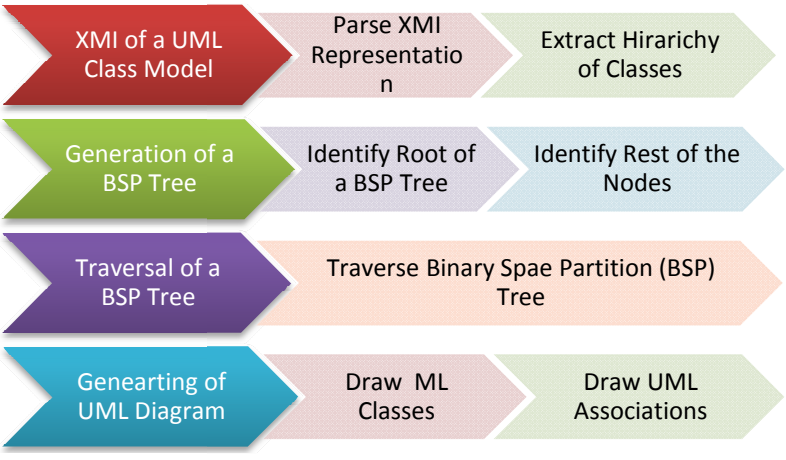

Fig. 1 Framework for Generation of UML class diagram from XMI

3.1 Parse XMI Representation

This module parses the given input (XMI representation of UML class model) using our XMI parser developed in VB.NET.

The XMI parser parses an XMI file with respect to target UML model by storing the data it in memory. For parsing XMI with VB.Net we used the `System.XML` namespace. The XML classes in the `System.Xml` namespace provide a comprehensive and integrated set of classes, allows working with XMI representation. These XML classes support parsing and writing XML, editing XML data in memory, and data validation. There are different classes available to read and write XML document in .Net Framework such as `XmlReader`, `XmlNodeReader`, `XmlTextReader`, and `XmlValidatingReader`.

A XMI parser is developed and used to parse XMI file to identify and extract class names, class attributes, class functions, association names, generalizations names, etc. While parsing an XMI presentation, XMI parser checks the syntax for well-formedness, and reports any violations (reportable errors). It helps to extract classes, objects and related tags.

The output of this phase is set of classes, their attributes, and methods, and all type of relationships such as associations, aggregations and generalizations.

3.2 Identifying Hierarchy of Classes

Once the information is extracted from XMI representation, next step is to identify relationships among multiple (two or more than two) classes and maps the associations and generalizations in classes and objects. Primary objective of this step is to extract possible hierarchy among all the classes. Such type of information helps in generating a BSP tree. Following rules are used to identify hierarchy among classes:

i. If a class *A* is associated to class *B*, class *B* will become child of class *A*.
ii. If a class *A* is generalization of another class *B*, class *B* will become child of *A* as *B* will be inheriting all features of *B*.
iii. If a class *A* aggregates another class *B*, class *B* will become child of *A*.
iv. If there is a class that has no relationship to other classes or there is numeration, which will be considered as leaves of a tree.

3.3 Generating a BSP Tree

We need to generate a BSP tree that can spatially distribute all classes into a diagram. By using the information (such as classes, associations, etc) extracted in section 3.1 and the hierarchal information identified in section 3.2, a BSP tree is constructed in this step. Then each class becomes a node of the root on the basis of the identified hierarchy. This process is recursively repeated in each half-space until every class has been incorporated into the tree. We have used the following algorithm to generate a BSP tree:

Step 1: Get a list of all the classes.

Step 2: Select a middle class *m*. Middle class *m* is selected as if total number of classes *n* is odd then $m = {}^{n}/_{2}$ else $m = {}^{n}/_{2} + 1$

Step 3: Put all of the classes before the class m, in XMI file, into a list *l* for left side of the BSP tree.

Step 4: Put all of the classes after the class m, in XMI file, into a list *r* for left side of the BSP tree.

Step 5: The class *m* simply becomes the root of the BSP tree. Thereafter, the classes in list *l* are placed in the BSP tree to the left of its parent class while the classes in list *r* are placed in the BSP tree to the right of its parent class.

Step 6: If list *l* and *r* has more elements, go to step2. Recursively repeat the process for both lists *l* and *r* until the last item in each list.

Once a BSP tree is generated, it is ready to be traversed and generate a UML class diagram. The traversal details of a BSP tree are given in next section.

3.4 Traversing the BSP Tree

To generate the UML class diagrams, we need to traverse the BSP tree, first. The tree is typically traversed in linear time from an arbitrary viewpoint. However, we are not drawing the UML model in a particular perspective of user's view. Hence, we do not consider the view point parameter here. Following algorithm was used for BSP tree traversal:

```
Function traverse(BSP_tree){
      if (BSP_tree != null){
         if (positive_side_of(root(BSP_tree))
            traverse(positive_branch(BSP_tree));
            display_triangle(root(BSP_tree));
            traverse(negative_branch(BSP_tree));
         else
            traverse(negative_branch(BSP_tree));
            display_triangle(root(BSP_tree));
            traverse(positive_branch(BSP_tree);
```

Fig. 2 Algorithm used for traversal of a BSP tree

In computer graphics, a BSP tree is in-order traversed. The process of in-order traversal recursively continues until the last node of the tree is traversed. In the case of a 2-*d* space tree, a modified in-order traversal (see Fig. 2) yields in a depth-sorted ordering of all rectangles (classes) in the space. Using the in-order traversal, either a back-to-front or front-to-back ordering of the triangles (classes) can be drawn. The back-to-front or front-to-back ordering is a matter of concern in the scenes where there are overlapping objects. However, in case of a UML class model, all objects (classes) in a scene are non-overlapping; the ordering of drawing does not matter.

3.5 Drawing UML Class Model

In this phase, first the extracted information from the previous module is used to draw the UML class diagrams. First of all the space is vertically divided from center. The class at root can be drawn at any side of the vertical division. Then each side (left & right) are horizontally and vertically drawn for classes represented by the child nodes. This process continues recursively until the last class is drawn.

Two physically draw class diagrams, a diagram generator was written in VB.NET using 2D Graphics library. Various graphics methods in GDI+ such as the DrawLine method draws a line between two points specified by a pair coordinates. DrawLines draws a series of lines using an array of points. We have also used GDI+ Graphics paths to redraw certain graphics items. The Graphics paths are used to handle multiple graphics items, including lines, rectangles, images, and text associated with a surface but we need to redraw only the rectangles. We can create a graphics path with all class diagrams (three rectangles) and just redraw that path, instead of the entire surface.

Once the rectangles for each class are drawn, each rectangle is filled with their respective attributes and methods. Then, the associations, aggregations and generalizations are drawn among the respective rectangles representing particular classes. The last step is to label the generated class diagrams with additional supportive information to make the class model more expressive.

4 Tool Support

The approach presented in section 4.2 was implemented in VB.NET as a proof of concept. A prototype tool was generated that can not only read XMI representation but also maps XMI information to a UML diagram. In XMI to UML class model mapping, a challenge was the implementation of BSP algorithm from generating UML classes as this has not been done before. The implementation of XMI2UML prototype tool consists of three key modules as below:

1. XMI Parser
2. BSP Tree Handler
3. Diagram Generator

Following is an overview of all these three modules those are counterpart of the XMI2UML prototype system. A screenshot of the implementation of XMI2UML tool in VB.NET is shown in Figure 3:

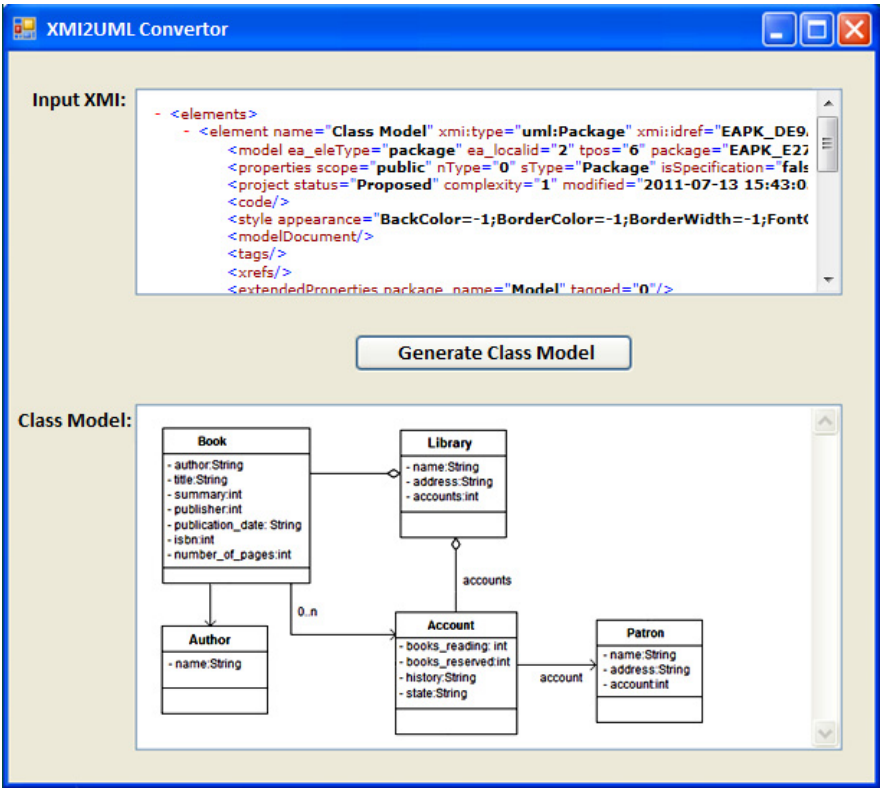

Fig. 3 Screenshot of XMI2UML prototype tool

4.1 XMI Parser

This module receives an XMI file and extracts all the UML class related information such as classes, attributes, methods, associations, etc. To read XMI representation arious classes are available in .Net Framework such as `XmlReader`, `XmlNodeReader`, `XmlTextReader`, and `XmlValidatingReader`. By the help of these classes, our XMI parser parses XMI file and extracts the required information.

4.2 BSP Tree Handler

The second module, BSP tree handler, receives output of the XMI parser and generates a BSP tree by using the algorithm discussed in section 4.2.3. In implementation of BSP tree generation algorithm, we used VB.NET ArrayList data structure.

4.3 Diagram Generator

The third and last module is diagram generator. To draw various shapes, we have used the .NET Framework Class Library that involves a whole host of classes given below:

- `System.Drawing`: Provides basic graphics functionality.
- `System.Drawing.Design`: Focuses on providing functionality for extending the design time environment.
- `System.Drawing.Drawing2D`: Provides two-dimensional and vector graphics classes and methods.
- `System.Drawing.Imaging`: Exposes advanced imaging functionality.
- `System.Drawing.Printing`: Gives you classes to manage output to a print device.
- `System.Drawing.Text`: Wraps fonts and type management.

All the above given classes are available in the `System.Drawing` namespace and its associated third-level namespaces. This module actually draws UML classes. One UML class was made by combining three rectangles into a polygon. All polygons were drawn on particular points spatially located by the BPS. Various functions in VB.NET such as `Graphics.FromHwnd()`, `Rectangle()`, `DrawRectangle()` were used to draw classes. Similarly, to draw associations, generalizations, the functions like `DrawLine()`.

5 Case Study

To test the designed tool XMI2UML, we present here a solved case study. The solved case study is a sub set of a Library Domain Model (LDM) [25] that describes main classes and relationships which could be used during analysis

phase to better understand domain area for Integrated Library System (ILS), also known as a Library Management System (LMS).

The problem statement consists of a UML Class model (see Figure 4) and a XMI 2.1 Representation. Both representations (UML and XMI) were generated using Enterprise Architect [24] v9.0 software. The original case study is available at [25]. However, we have solved a subset of the case study. The solved subset consists of five classes and five relationships. Fig 4 shows the subset (UML class model) of the solved case study. Following is the problem statement for the case study (See Figure 4):

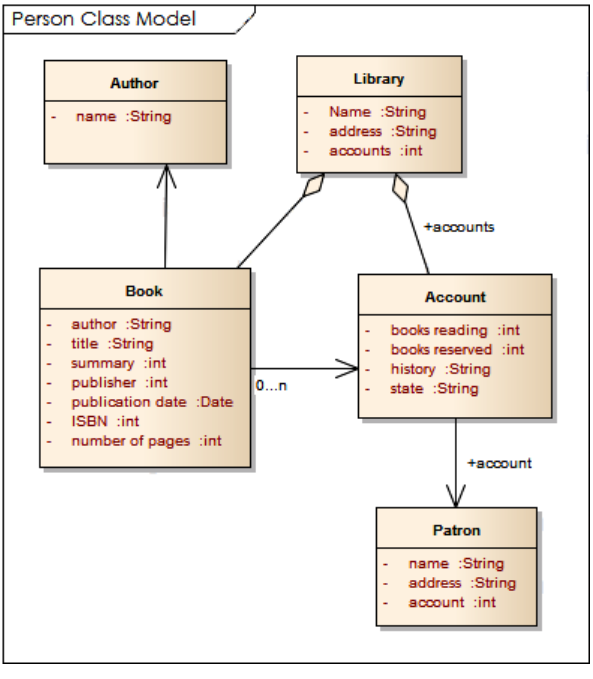

Fig. 4 UML class model of the Case Study problem statement

The problem statement of the case study was given as input (XMI representation) to the XMI2UML tool. The tool parses the input and extracts the UML class elements (see Table 1):

Table 1 Extracted UML Class Elements

Category	*Count*	*Details*
Classes	05	Author, Book, Library, Account, Patron
Attributes	18	name, author, title, summary, publisher, publication date, ISBN, number of pages, name, address, accounts, books reading, books renewed, history, state, name address account
Operations	00	-
Objects	00	-
Multiplicity	01	0..1
Associations	03	Account
Aggregations	02	Accounts
Generalizations	00	-

Once all the UML Class elements were extracted, a logical model of the target class diagram was developed. The logical model was based on relationships in all candidate classes. The generated logical model is shown in Table 2:

Table 2 Identifying relationships in a UML class model.

S# / Type	*Source*	*Mult. A*	*Label*	*Mult. B*	*Destination*
Relation 01	Book	-	-	-	Author
Relation 02	Book	-	-	-	Library
Relation 03	Book	0...n	-	-	Account
Relation 04	Account	-	Accounts	-	Library
Relation 05	Account	-	Account	-	Patron

Once the relationships in a UML class model are generated, the graphical module of XMI2UML module generates the target UML class diagram. The finally labeled UML class diagram is shown in Figure 5 where the orange doted lines are showing the binary partition of the space:

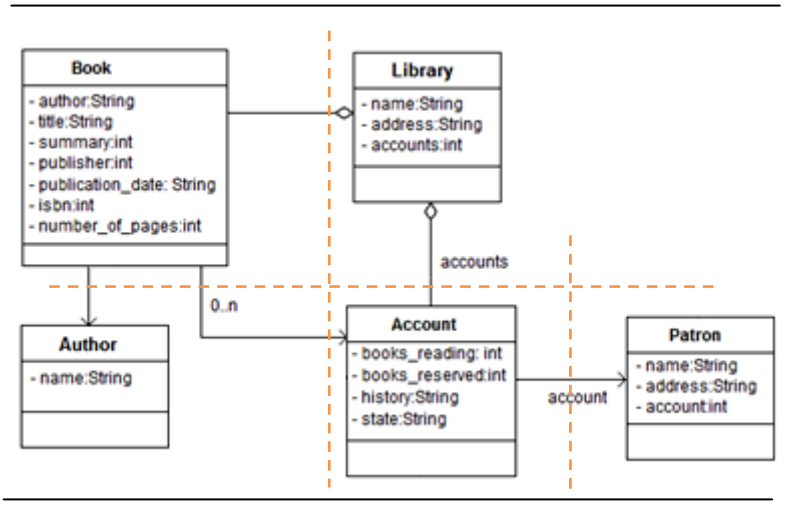

Fig. 5 Finally generated UML class model

There is an issues regarding efficiency of a BSP tree that relates to the balancing of the tree. Regarding balancing of a BSP tree, Bruce Naylor has presented some good algorithms and techniques to maintain well-balanced BSP trees. For a bigger UML class models, where we have to deal with bigger BSP trees, well-balancing of a tree become more critical.

6 Conclusions

The This research is all about designing and implementing a theory that can read, understand and analyze the XMI file and generate UML class models. The proposed system will be fully automated and able to find out the classes and objects and their attributes and operations using an artificial intelligence technique. Then the UML diagrams such as class diagrams would be drawn. The accuracy of the software is expected up to about 80% without any involvement of the software engineer provided that he has followed the pre-requisites of the software to prepare the input scenario. The given input should be an XMI file. A graphical user interface is also provided to the user for entering the Input XMI representation in a proper way and generating UML diagrams. The current version of XMI2UML tool can process XMI 2.1 version. This research was initiated with the aims that there should be software which can read the XMI representation and can draw the UML class diagrams

References

[1] OMG. XML Metadata Interchnage (XMI) version 2.1. Object Management Group (2005), http://www.omg.org

[2] OMG. Unified Modeling Language: Diagram Interchange version 2.0, OMG document ptc/03-07-03 (July 2003), http://www.omg.org

[3] Alanen, M., Lundkvist, T., Porres, I.: A Mapping Language from Models to XMI (DI) Diagrams. In: Proceedings of the 31st Euromicro Conference on Software Engineering and Advanced Applications, pp. 450–458. IEEE Computer Society, Portugal (2005)

[4] Bailey, I.: XMI, UML & MODAF (February 2005), http://www.modaf.com

[5] Dolog, P., Nejdl, W.: Using UML and XMI for Generating Adaptive Navigation Sequences in Web-Based Systems. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 205–219. Springer, Heidelberg (2003)

[6] Kovse, J., Härder, T.: Generic XMI-Based UML Model Transformations. In: Bellahsène, Z., Patel, D., Rolland, C. (eds.) OOIS 2002. LNCS, vol. 2425, pp. 192–198. Springer, Heidelberg (2002)

[7] OMG: Meta Object Facility Specification (MOF) version 1.3.1, Object Management Group (2001), http://www.omg.org

[8] Wagner, A.: A pragmatically approach to rule-based transformations within UML using XMI. In: Workshop on Integration and Transformation of UML Models (2002)

[9] Jensen, M.R., Møller, T.H., Pedersen, T.B.: Converting XML Data To UML Diagrams For Conceptual Data Integration In: 1st International Workshop on Data Integration over the Web (DIWeb) at 13th Conference on Advanced Information Systems Engineering, CAISE 2001 (2001)

[10] Mich, L.: NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. Natural Language Engineering 2(2), 167–181 (1996)

[11] Delisle, S., Barker, K., Biskri, I.: Object-Oriented Analysis: Getting Help from Robust Computational Linguistic Tools. In: 4th International Conference on Applications of Natural Language to Information Systems, Klagenfurt, Austria, pp. 167–172 (1998)

[12] Börstler, J.: User - Centered Requirements Engineering in RECORD - An Overview. In: Nordic Workshop on Programming Environment Research, NWPER 1996, Aalborg, Denmark, pp. 149–156 (1999)

[13] Overmyer, S.V., Rambow, O.: Conceptual Modeling through Linguistics Analysis Using LIDA. In: 23rd International Conference on Software Engineering (July 2001)

[14] Perez-Gonzalez, H.G., Kalita, J.K.: GOOAL: A Graphic Object Oriented Analysis Laboratory. In: 17th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2002, NY, USA, pp. 38–39 (2002)

[15] Harmain, H.M., Gaizauskas, R.: CM-Builder: A Natural Language-Based CASE Tool for Object- Oriented Analysis. Automated Software Engineering 10(2), 157–181 (2003)

[16] Oliveira, A., Seco, N., Gomes, P.: A CBR Approach to Text to Class Diagram Translation. In: TCBR Workshop at the 8th European Conference on Case-Based Reasoning, Turkey (September 2006)

[17] Anandha Mala, G.S., Uma, G.V.: Automatic Construction of Object Oriented Design Models [UML Diagrams] from Natural Language Requirements Specification. In: Yang, Q., Webb, G. (eds.) PRICAI 2006. LNCS (LNAI), vol. 4099, pp. 1155–1159. Springer, Heidelberg (2006)

[18] Bajwa, I.S., Samad, A., Mumtaz, S.: Object Oriented Software modeling Using NLP based Knowledge Extraction. European Journal of Scientific Research 35(01), 22–33 (2009)

[19] Boger, M., Jeckle, M., Müller, S., Fransson, J.: Diagram Interchange for UML. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 398–411. Springer, Heidelberg (2002)

[20] Ranta-Eskola, S., Olofsson, E.: Binary Space Partioning Trees and Polygon Removal in Real Time 3D Rendering. Uppsala Master's Theses in Computing Science (2001)

[21] Fuchs, H., et al.: Near Real-Time Shaded Display of Rigid Objects. Computer Graphics 17(3), 65–69

[22] Ize, T., Wald, I., Parker, S.: Ray Tracing with the BSP Tree. In: IEEE Symposium on Interactive Ray Tracing, RT 2008, pp. 159–166 (2008)

[23] Chen, S., Gordon, D.: Front-to-Back Display of BSP Trees. IEEE Computer Graphics & Algorithms, 79–85 (September 1991)

[24] Matuschek, M.: UML - Getting Started: Sparx Enterprise Architect, To a running UML-Model in just a few steps, by Willert Software Tools (2006), http://www.willert.de/assets/Datenblaetter/UML_GettingStarted_EA_v1.0en.pdf

[25] Library Domain Model (LDM), http://www.uml-diagrams.org/class-diagrams-examples.html

[26] Bajwa, I.S., Amin, R., Naeem, M.A., Choudhary, M.A.: Speech Language Processing Interface for Object-Oriented Application Design using a Rule-based Framework. In: International Conference on Computer Applications, ICCA 2006, pp. 323–329 (2006)

[27] Bajwa, I.S., Choudhary, M.S.: Natural Language Processing based Automated System for UML Diagrams Generation. In: Saudi 18th National Conference on Computer Application, NCCA 2006, pp. 171–176 (2006)

[28] Bajwa, I.S., Naeem, M.A., Ali, A., Ali, S.: A Controlled Natural Language Interface to Class Models. In: 13th International Conference on Enterprise Information Systems, ICEIS 2011, Beijing, China, pp. 102–110 (2011)