Computer Assisted 2-D Graphical Software Modelling using BSP Tree Algorithm

Imran Sarwar Bajwa, Mamoona N. Asghar, Muhammad Anwar Shahzada

***Abstract*— Current approaches/CASE Tools are unable to provide support for interchanging Meta data of models in the form of XMI. Due to lack of this support research of current approaches /Tools for automated software model cannot be incorporated in existing state of Art CASE tools. Even after providing the support of Meta data interchanging support using XMI in existing approach cannot automatically generate graphical model. Due to absence of any algorithm support for generation of graphical model. One of the reasons the current CASE tools are unable to import/export graphical models is inability of XMI to store 2-D spatial information regarding coordinates of the elements of a graphical model as XMI can inherently stores just metadata of a graphical model. This paper presents a novel idea of using Binary Space Partitioning (BSP) tree for diagram generation as BSP is an efficient spatial data structure. The used approach cannot only perform a set of Boolean operations on complex graphical models but can also perform 2-D space partitioning. A Java based implementation as Eclipse plugin is also discussed in this paper. The results of experiments express that the used approach is efficient and accurate.**

***Index Terms*— Software Class Models, Unified Modelling Language, 2-D Space Partitioning, BSP Algorithm**

I. INTRODUCTION

Since last couple of decades, graphical modeling is very popular for graphically elaborating software models (such as structural models, behavioural models, start charts, etc.) in the domain of software engineering. UML (Unified Modeling Language) is a standard used to seamlessly design graphical models of complex and large-sized software systems [1]. Various activities performed at different stages of Software Development Life Cycle (SDLC) are model generation, models' consistency-analysis, design pattern generation, code engineering, reports and documentation preparation, etc. UML based graphical models are part of almost all of these SDLC activities. Various Computer Assisted Software (CASE) tools are used to perform the above mentioned. Typically, in a software development team, team members have to share the artefacts designed in various CASE tools used at different stages of SDLC. Such CASE tools assist in accelerating the development of a project by simplifying various stages of SDLC.

Typically used CASE tools are MS Visio, Rational-Rose, ArgoUML, Altova, Enterprise-Architect, Smart-Draw, etc. provide facility of importing and exporting metadata of a software model using XMI format 1.0 or 2.0. XMI is an OMG's standard used for metadata interchange from one platform to others [2, 3, 4, 5]. However, none of the tools has ability of re-generating visual artefacts from XMI except Enterprise Architect [6] that can just generate diagrams just from its own generated XMI. A key reason of this non-conformity in XMI and graphical models is that XMI does not store any graphical information that can help in drawing a model element along 2-D spatial coordinates and avoiding any possible overlying of elements of a visual artefact. A solution of this problem is presented in [19] by proposing additions in XMI metamodel to address this problem to store graphical coordinates information. Though, such additions in XMI metamodel can become a performance overhead since resultant XMI will be more complex to machine process. Due to this reason, import/export of a complete graphical model (such as a UML class model or a BPML process model) designed in a CASE tool is a thought-provoking task, to date. A novel idea is proposed in this paper that automatically generates a UML artefact such as a UML class model from XMI with the help of BSP (Binary Space Partitioning) tree [7] data structure and algorithm. In various applications of computer graphics, graphical information of a 2-D diagram can be captured in a BSP Tree [8]. A BSP tree can be useful in capturing the spatial arrangement and relations in a set of objects of a UML class diagram. After capturing the graphical coordinates, a parse tree is traversed to re-generate the same UML class model by efficiently partitioning the 2-D space. Once all the UML classes are drawn on a 2-D space, extra information is tagged such as various relationships between the target elements e.g. aggregations associations, generalisations, etc.

In remaining paper, Section 2 discusses the BSP tree data structure used for 2-D space portioning. Section 3 describes the used algorithm for space portioning and drawing model. An example of experimentation is presented in section 4 discusses evaluation details. An account of related work is given in section 5 and finally, paper is concluded.

II. MATERIAL AND METHOD

Famous geometrical problems such as ray-tracing [9], Constructive solid geometry (CSG) [10, 11], etc. have been successfully solved by using BSP tree data structure. Fuchs, et al. presented BSP that partitions a 2-D space having a set of polygons [12]. The BSP tree works by splitting a 2-D space into couple of partitions as dimension d-1. The process of generating binary partitions is repeated unless the total space is segregated into smaller portions called "cells". However, when a BSP tree is generated having graphical information of a model or a diagram, the tree can be traversed easily to re-produce the model. We aim to use BSP trees in automatic generation of UML diagram from XMI. We have performed this process into two phases:

Imran Sarwar Bajwa, Mamoona N. Asghar, Muhammad Anwar Shahzada, Department of Computer Science, The Islamic University of Bhawalpur. Email: Imran.sarwar@iub.edu.pk. Manuscript received July 13, 2016; revised on Aug 17, 2016; accepted on Sep 26, 2016.

 ISSN – 1999-4974

A. Generating of a BSP Tree

The initial step is generation of a BSP tree of elements of a UML artefact such as a class diagram. In generation of a BSP tree, it is ensured that a BSP-tree has all the related information about UML class model. We can identify hierarchy of classes with the help of an algorithm given below:

TABLE. 1 ALGORITHM FOR IDENTIFYING CLASS HIERARCHIES

Algorithm 1: Identification of Hierarchy of classes
Require: List of classes

1: **If** class-A is associated with class-B **then**
class-B will be a child of class-A
2: **If** class-A has generalization of other class-B **then**
class B will becomes a child of class-A(a class-B will inherit all features of Class-A)
3: **If** class-A is aggregates another class-B **then**
class-B will becomes child of class-A.
4: **If** class-A has composition of another class-B **then**
class-A will becomes child of class-B.
5: **If** class-A has realization with an interface B **then**
class A will become child of B.
6: **If** class A has no relationship to any other classes or there is numeration **then**
class A is being considered as leaves of a tree.

BSP tree not only has information about number of classes as well as hierarchical structure of elements in a UML artefact. It's better to generate a Binary space partitioning tree from XMI file rather than to generate Binary space partitioning tree through graphical scene. For example, all classes are C1, C2, …, Cn in an XMI document are extracted from XMI document and stored in 1-D array C. First we need to find out the hierarchy in which these classes are going to be stored in a BSP tree data structure.

We have used such associations to find the hierarchy among the nodes in a tree. The used algorithm for generating BSP tree for UML class diagram is given below:

TABLE. 2 ALGORITHM FOR GENERATING BSP TREE

Algorithm 2: For Generating BSP Tree
Require: List of all classes

1: Select a middle class C_m. Middle class C_m is any of the classes that have exactly two relations.
2: Put all the classes in XMI file before the class C_m, into a list $C_{0\ldots m-1}$ on Left-Side of a BSP data structure.
3: A set of classes which exist after the class C_m are placed in XMI into a list $C_{m+1\ldots n}$ to be right-side of the Binary Space Partitioning tree.
4: The class C_m will become the root of the Binary Space Partitioning tree. After that, the available classes in a list from $C_{0\ldots m-1}$ are put in the Binary Space Partitioning data structure to the left-side of the super class while the remaining set of classes are kept in list from $C_{m+1\ldots n}$ are arranged in the Binary Space Partitioning tree to the right-side of its super class.
5: If in a list $C_{0\ldots m-1}$ and $C_{m+1\ldots n}$ having more elements, go back to step No.2. By applying Recursive process again and again on the lists $C_{0\ldots m-1}$ and $C_{m+1\ldots n}$ till the last element in the list.

To generate a tree the first requirement is to identify the suitable class for root node. A class having exactly two of relationships (identified by Algorithm 1) becomes C_m that becomes the root node C *root* of the BPS tree. All the remaining classes in array C will become child nodes of the root in the tree. Here, an associated class C*a* will becomes a child of class C *root*. Here, all classes on the left side of the root node are C^l and can be find out by using function $f(C^l) < C$ *root*. Similarly, all classes on right side of the root are C^r and function to find these classes is $f(C^r) > C$ *root.* Here, association in two classes is determined on the basis of UML associations, UML generalizations and UML aggregations.

B. Traversing of BSP Trees

After generating a BSP tree, next phase is its traversing for the sake of 2D space portioning. During the space portioning process, coordinates of each partition are also saved that are later on used for drawing graphical shapes to generate UML class diagrams.

TABLE. 3 ALGORITHM FOR TRAVERSING BSP TREE

Algorithm 3: BSP Tree Traversal

Require: A BSP tree
1: **function***Traverse1*(BSPtree)
2: **if** (BSPtree==null)**then**
3: exit
4: **end if**
5: **if** (positivesideof(rot(BSPtree))
6: traverse1(positivebranch(BSPtree));
7: displaytriangle(rot(BSPtree));
8: traverse1(negativebranch(BSPtree));
9: **else**
10: traverse1(negativebranch(BSPtree));
11: displaytriangle(rot(BSPtree));
12: traverse1(positivebranch(BSPtree);
13: **end if**
14: **end function**

To traverse a BSP tree, a simple test is performed at each node of the tree recursively. In the algorithm for BS tree traversal shown in Table III, a slightly adapted version of traditional in-order traversal algorithm [13] is used. The process of traversing a BSP tree and its mapping to a binary partitioned space is shown in Figure 1. A major modification in [13] is not using the view point parameter as the originally this algorithm was proposed for rendering purpose.

In our approach, to draw a UML class model, a 2-D space is portioned into a set of cells as shown in Figure 1 and a rectangle is drawn to represent a cell. Since, each partition is a binary partition, each rectangle has a neighbouring rectangle by side and may be a following rectangle as well.

ISSN – 1999-4974

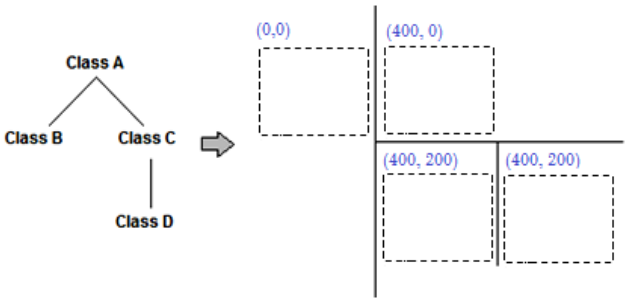

Fig. 1 BSP tree to UML class model mapping

C. Used Framework for Diagram Generation

The process of generating diagram is performed into two phases: initially, XMI script of a UML class diagram is mapped to a BSP tree and then by traversing the BSP tree re-generation of a UML class model is possible. In our approach, the experiments were performed with 2.1 version of XMI and the reason of choosing 2.1 version of XMI was acceptance by most of the major CASE tools e.g. Enterprise Architect, ArgoUML, Altova UModel, USE, etc.

The proposed framework is shown in Figure 2 where input of the system is XMI (.xml) script and then a set of processing activities are executed to perform the steps explained in the previous section.

All the steps of proposed framework for 2-D space partitioning and UML class diagram generation are explained below:

D. Parsing XMI Script

The working of the used framework starts with input acquisition of metadata of a UML class diagram in the form of XMI. We have used a XMI parser developed in Java as Eclipse plugin for this purpose. Since, we use the EMF (Eclipse Modeling Framework) Ecore format of XMI, the parsing module is based on Ecore parsing libraries. During parsing of Ecore XMI, the names of classes, their respective attributes and methods are extracted along the various types of relationships. It helps to extract classes, objects and related tags. The extracted information is stored into an **`ArrayList<String>`** as class names, respective class variables, and class functions along associated relationships.

E. Classes Hierarchy Identification

After processing the XMI script for information processing, the associations in classes is extracted. Here, different associations are represented in a different way. Extraction of such associations is also important as the hierarchy of nodes in a BSP tree representing classes is based on the types of relationships such as associations, aggregations, etc.

F. BSP Tree Generation

Once the required information such as class names, attributes, associations, etc. is retrieved, the BSP tree is generated. To generate a BSP tree, first of all a root node is selected by randomly choosing a class that comes first in XMI script will also be first class in **`ArrayList<String>`**. This process of iteratively repeated by selecting the next class in **`ArrayList<String>`** and deciding its hierarchy by using algorithm 1 and then generating next node of the BSP tree by using algorithm 2. This process is repeated recursively until every polygon is marked as a node of the BSP tree.

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.

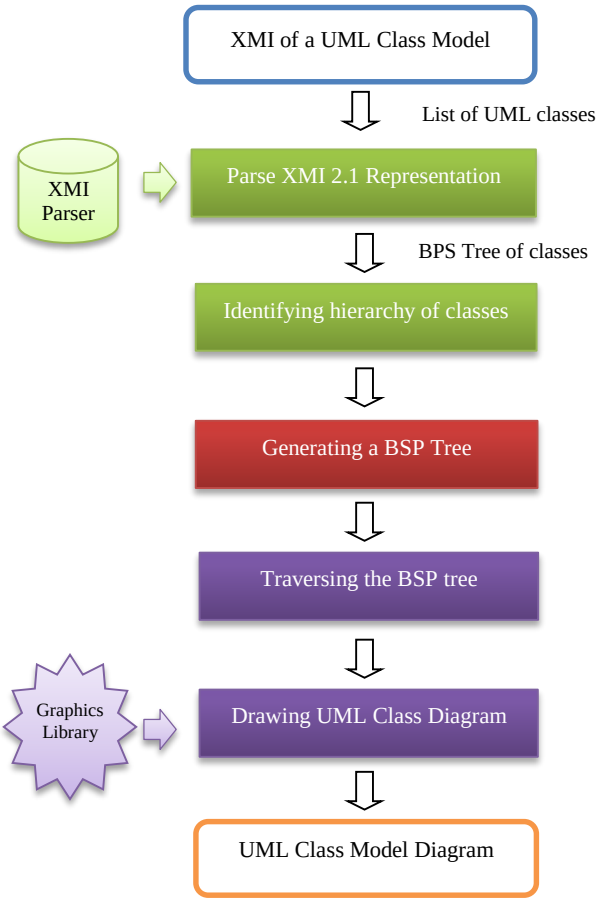

Fig. 2 Framework of the used system for 2-D space partitioning

G. 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.

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. 1) 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.

ISSN – 1999-4974

H. Drawing UML Class Model

In this phase, first the extracted information from the previous module is used to draw the UML class diagrams.

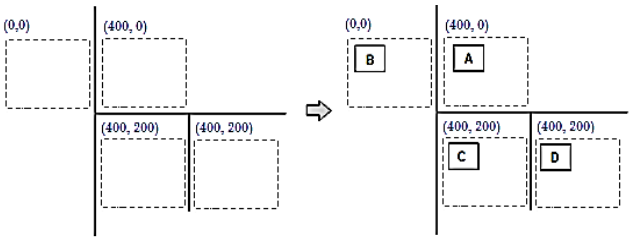

Fig. 3 Mapping the BSP tree to UML class Diagram

For the root node, the whole 2-D space is bisected into vertical position. Here, the first class that is root node can be drawn at any side of the partition however our algorithm draws the class on the left side of the vertical division. To actually draw class model, a Java based module was designed that is based in 2D Graphics library that can draw rectangles, lines on given coordinates. The rendering process in Java can be broken down into various phases that are controlled by the Graphics 2D rendering attributes [28].

To draw one a class diagram, a combination of three rectangles are drawn on a JPanel so that the size of the rectangle may be adjustable. Here, all three rectangles are filled with class names and other information. Afterwards the relationships are also drawn on 2-D space. Finally, the diagram is labeled

III. RESULTS AND DISCUSSION

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) [18] 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 Fig 4). Here EMF Ecore format of XMI is used and the Ecore of UML class model shown in Figure 4 is designed in Eclipse EMF. The used example in the experiment (see Figure 4) is given in [18].

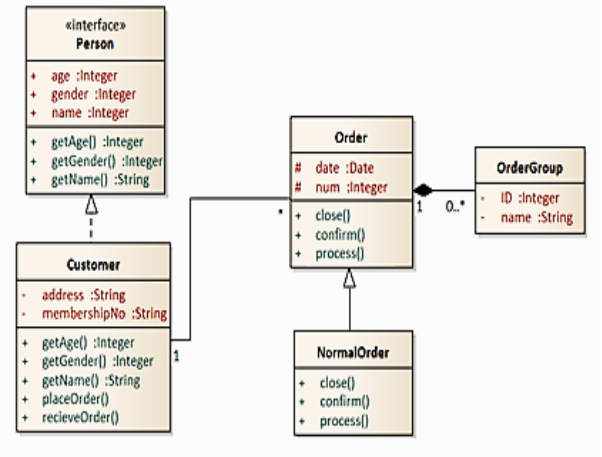

Fig. 4 Example of EMF Ecore XMI

Our tool XMI2UML tool takes input EMF ECore XMI of the UML class model shown in figure 4 and then the proposed tool processes data and generated output is shown in Table IV and the extracted relationships are used in Table V:

TABLE. 4 OUTPUT OF EMF ECORE XMI

Category	*Count*	*Details*
Classes	05	Person, Customer, Order, NormalOrder, OrderGroup
Attributes	09	age, gender, name, address, membershipno, date, num, id, name
Operations	00	getAge(), getGender(), getName(),getAge(), getGender(), getName(), placeOrder(), recieveOrder(), close(), confirm(), proceed(),close(), confirm(), proceed(),
Objects	00	-
Multiplicity	02	*, 0…*
Associations	01	Order – Customer
Composition	01	Order – OrderGroup
Aggregations	00	-
Generalizations	01	Order - NormalOrder
Realization	01	Customer -- Person

After extracting the information of a UML Class model, a logical model of a BSP tree data structure is generated as shown in Figure 5.

TABLE V RELATIONSHIP EXTRACTION FROM EMF ECORE XMI

S No./ Type	*Source*	*Mult. A*	*Label*	*Mult. B*	*Destination*
Relation 01	Order	*	-	1	Customer
Relation 02	Order	1	-	0 … *	OrderGroup
Relation 03	Order	-	-	-	NormalOrder
Relation 04	Customer	-	-	-	Person

ISSN – 1999-4974

The BSP tree generated for the solved example is shown in Fig. 5.

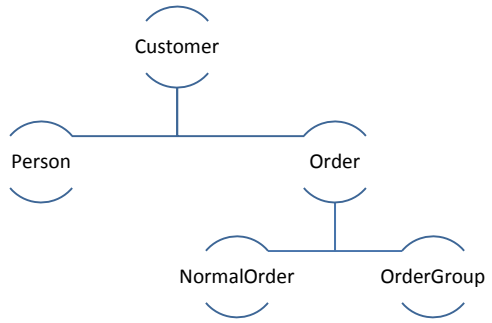

Fig. 5 Binary Space Partition Tree generated for the example

After generating a logical model based BSP tree the BSP tree is mapped to an actual diagram and the actual diagram of the solved case study above is shown in in Figure 6. In the Figure 6 the coloured dotted lines show the rationing done by the BSP tree algorithm.

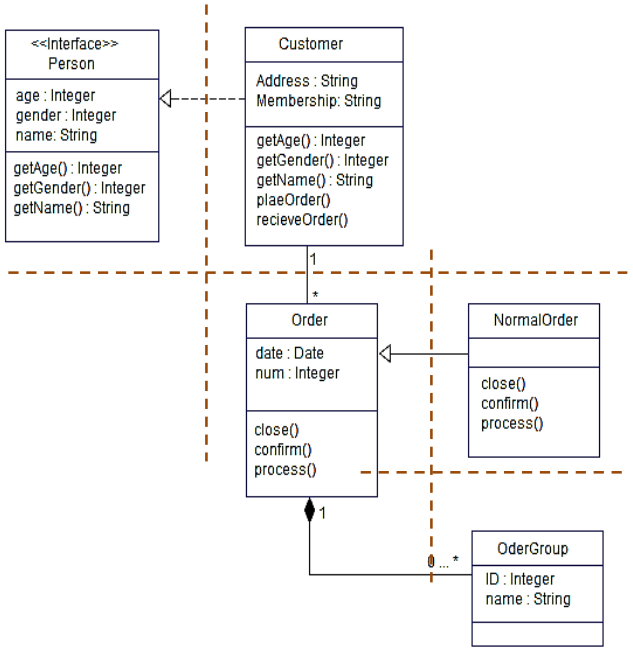

Fig. 6 output of XMI2UML tool

Different examples were solved by the XMI2UML tool and the results are shown in Table VI. However, an efficiency related issue of BSP that was highlighted by Bruce Naylor is balancing of the BSP tree. To solve this problem various tree balancing algorithms can be used. Including the examples discussed above; total 5 different scenarios were processed. The results of all examples are shown in Table VI.

TABLE VI RESULTS OF DRAWN UML CLASS MODELS

Category	Total Items	Correct Items	Missed Items	Incorrect Items
Classes	36	30	2	3
Attributes	36	33	1	2
Relationships	20	16	1	3
Cardinality	20	18	0	2

The results of the experiments are further represented in terms of recall and precision. Table VII shows that recall of Classes is 86.11% and precision is 91.71% that is very encouraging for future enhancements (see Fig. 7.).

TABLE VII RECALL AND PRECISION OF RESULTS

Category	Recall	Precision
Classes	86.11%	91.17%
Attributes	91.66%	93.93%
Relationships	80.33%	84.21%
Cardinality	90.00%	90.90%

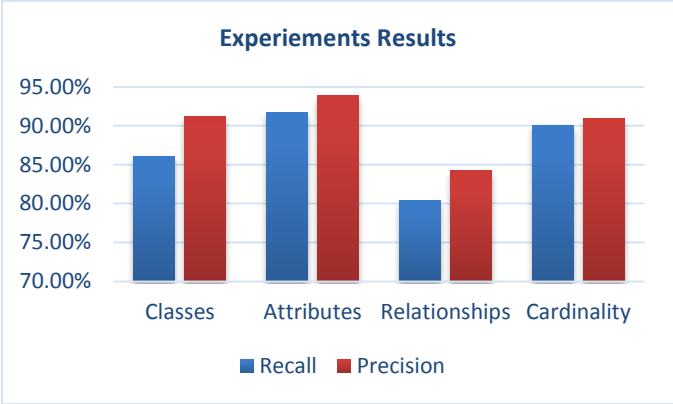

Fig. 7 Manifestation of results

Figure 7 shows the results of the experiment as the graph shows that accuracy of classes is 91% and attributes is 94% while accuracy is of relationships and cardinality is at lower side as 78% and 90% respectively.

IV. CONCLUSION

In this research paper, a challenging task of automated creation of UML class diagram with software requirements is addressed since natural language requirements do not have a particular graphical representation. We integrated BSP trees for automated generation of UML class models. Our research is all about designing and implementing a theory that can read, understand and analyze the natural language software requirements in text file and transform into XMI format and then we generated UML class diagram from XMI. We have designed a fully automated framework which has an ability to identify classes, class objects, attributes, Associations, relations among classes and operations by applying an artificial intelligence based methodology. Class diagrams are generated by using the knowledge which is extracted through framework. Even there is no involvement of user normal

ISSN – 1999-4974

accuracy of the product is more than 81%. The software developers input software requirements specifications in Text file and it generate class diagram. We also provided an interface in graphical form through which a user can enter text file of software requirements and then UML class diagram is generated from the input text.

REFERENCES

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

[2] OMG. (2005). XML Metadata Interchange. (XMI) version 2.1. Object Management Group. Available: http://www.omg.org.

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

[4] Malesevic, A., Brdjanin, D., & Maric, S. (2013, July). Tool for automatic layout of business process model represented by UML activity diagram. InEUROCON, 2013 IEEE (pp. 537-542). IEEE.

[5] Kovse, J., Härder, T., (2002). Generic XMI-Based UML Model Transformations. OOIS '02 Proceedings of the 8th International Conference on Object-Oriented. September 2002.

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

[7] Ferdowsi, S., Voloshynovskiy, S., Gabryel, M., & Korytkowski, M. (2014, January). Multi-class Classification: A Coding Based Space Partitioning. InArtificial Intelligence and Soft Computing (pp. 593-604). Springer International Publishing.

[8] Hertzog, P. (2015, March). Binary Space Partitioning Layouts to Help Build Better Information Dashboards. In Proceedings of the 20th International Conference on Intelligent User Interfaces (pp. 138-147). ACM.

[9] T. Ize, I. Wald, and S. Parker. 2008. Ray Tracing with the BSP Tree. In IEEE Symposium on Interactive Ray Tracing (RT'08), pp.159-166, 2008.

[10] C. Segura, T. Stine, J. Yang. (2013). Constructive Solid Geometry Using BSP Tree. Computer-Aided Design, 24-681: Availableat:https://www.andrew.cmu.edu/user/jackiey/resources/CSG/CSG_report.pdf

[11] Veeramani, A., Venkatesan, K., & Nalinadevi, K. (2014, October). Abstract Syntax Tree based Unified Modeling Language to Object Oriented Code Conversion. In Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing (p. 25). ACM.

[12] Badreddin, O., Lethbridge, T. C., & Forward, A. (2014, January). A novel approach to versioning and merging model and code uniformly. In Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on (pp. 254-263). IEEE.

[13] Park, J., Chun, I., Hong, S. H., Yoon, T., & Kim, W. (2014). A Resource Monitoring Code Generation for Self-healing. International Information Institute (Tokyo). Information, 17(3), 1097.

[14] Satish, P., Paul, A., & Rangarajan, K. (2014, March). Extracting the combinatorial test parameters and values from UML sequence diagrams. In Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on (pp. 88-97). IEEE.

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

[16] Marko Boger, Mario Jeckle, Stefan Mueller, Jens Fransson. 2002. Diagram Interchange for UML. UML 2002: 398-411

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

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

[19] Docs.oracle.com, (2015). Graphics2D (Java Platform SE 7). Availableat:http://docs.oracle.com/javase/7/docs/api/java/awt/Graphics 2D.html [Accessed 7 Jan. 2015].

[20] Hameed, K., & Bajwa, I. S. (2012). Generating Class Models Using Binary Space Partition Algorithm. In Computer and Information Science 2012 (pp. 1-13). Springer Berlin Heidelberg.

ISSN – 1999-4974