Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 10 (17): 1894-1901, 2010
ISSN 1812-5654

Graph Drawing Algorithms: Using in Software Tools

Hamid Sanatnama and Farshad Brahimi
Faculty of Mathematics and Computer Science, Shahid Bahonar University of Kerman, Kerman, Iran

Abstract: The key understanding of a complex system is viweing it's different levels of abstraction. Visualization of the architectural structure of such systems will increase our perception of a complex system. There are many Computer-Aided Software Engineering (CASE) Tools for visual developing. In software engineering visualization and graphic-oriented methods are very common and in most cases these methods use 2D graphics, but 3D software models enable visualization of richer semantics than those of 2D models. This study reviews some of available algorithms with focus on drawing class diagram. We also presented some of the developed tools with their domain of use. We believe that this review is a step towards to investigate the capability of these algorithms for visualization of interactions between components composed by Component Interaction Markup Language (CIML). CIML is well-formed framework for describing the components and the interaction between them.

Key words: UML, graph drawing, class diagram, CASE tools

INTRODUCTION

One of the major problems software engineers are facing is that software can be extremely complex. Source code shows us extreme detail and a little about the operation of the system as whole. The key understanding such a complex system is being able to view it at different levels of abstraction. Visualizing architectural structure will increase our perception of a complex system.

The two worlds have been brought together in this paper; UML-Unified Modeling Language as a standard of visual design of software application and Automatic layout of software diagrams which has been an attractive graph drawing application for use in software tools. The problem is after adding an object or a relation between them the graph layout tool could automatically re-position the objects and lines so that the diagram is more comprehensible. We believe that this review is a step towards to investigate the application of these algorithms for visualization of interactions between components composed by Component Interaction Markup Language (CIML) presented by Sanatnama *et al*. (2009).

UML: The first version of UML was created by three amigos-Grady Booch, Ivar Jacobson and Jim Rambaugh from Rational Software. UML is used for a visual development where some steps of a developing process are performed by designing diagrams representing application parts. The main advantage of visual development is easier understanding and orientation in designed diagrams. For purpose of visual development various UML CASE-Computer-Aided Software Engineering Tools exits. The main feature of UML CASE tools are design UML diagrams.

Class diagram: The visual specification of types of objects that exist in a system and the relationships that exist among them are class diagrams. It is used as a basis to develop other UML diagrams including sequence and collaboration diagrams. Class diagrams represent structural and not behavioral relationships that exist among system entities. There are two main categories of relationships between classes as shown in Fig. 1; inheritance, associations.

Class diagrams have been used frequently to visualize the static structure of object-oriented programs. The UML is agreed by the software engineering community in the last years as a common national standard (OMG, 2001).

Graph drawing: The fact is that most diagramming methods are based on graphs. For example, in flow charts the graph nodes are computational steps and edges represent control flow. Similarly, in E-R and class diagrams nodes are entities and edges are the relations between them, etc.

In software engineering visualization and graphic-oriented methods are very common and in most cases these methods use 2D graphics. The use of 2D drawings is inherited from traditions in engineering, where they are

Corresponding Author: Hamid Sanatnama, Faculty of Mathematics and Computer Science, Shahid Bahonar University of Kerman, Kerman, Iran Tel/Fax: +981-341-3220059

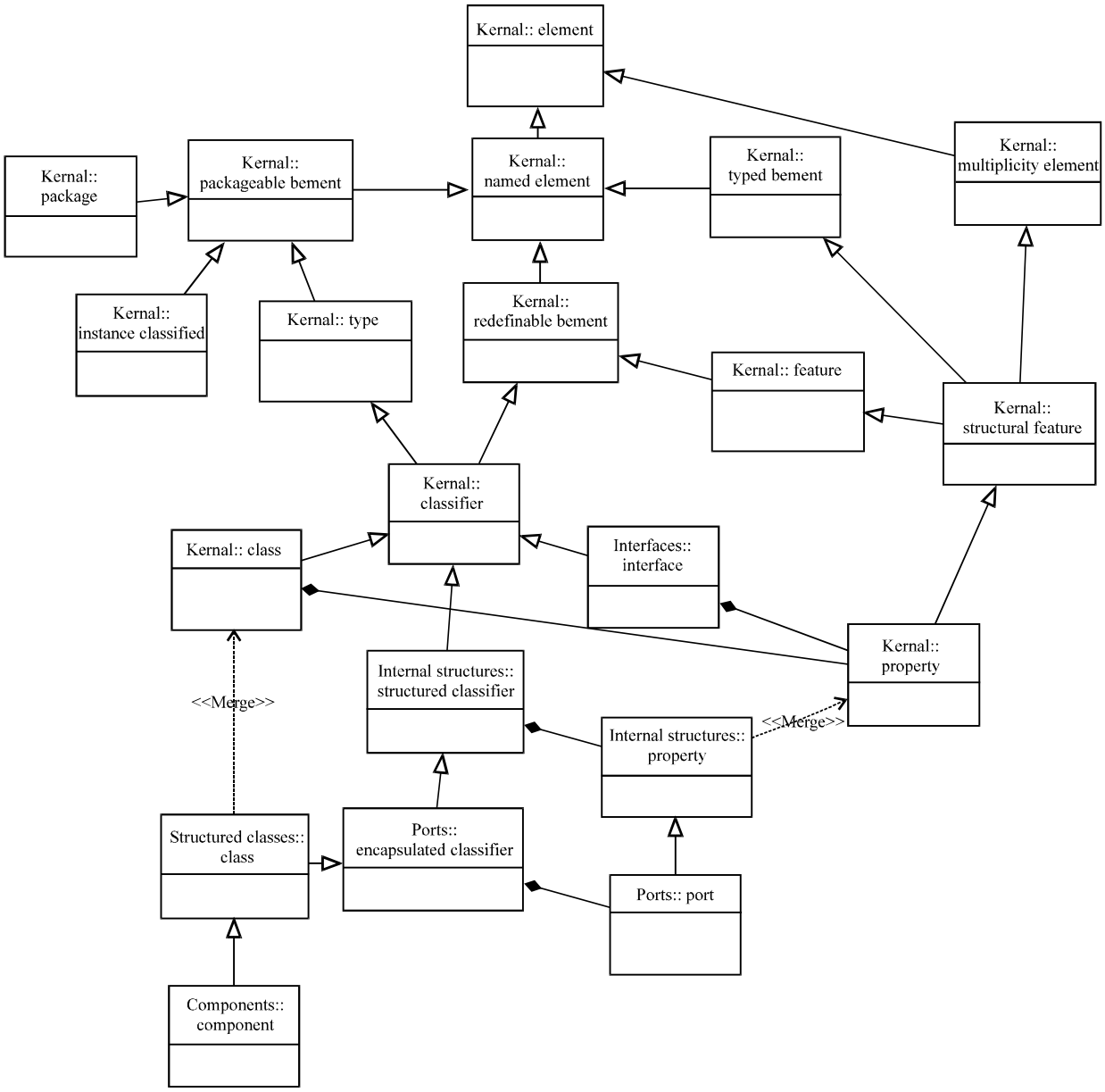

Fig. 1: An example of UML class diagram

working well. The great advantage of 2D drawings is that they only require paper and pencil. On the other hand 3D software models enable visualization of richer semantics than those of 2D models. For example, there are 3D models that encompass both a static and a dynamic description of a system.

In the rest of this paper we will consider the following graph drawing algorithms: (Sugiyama *et al*., 1981; Seemann, 1997; Purchase *et al*., 2001; Klaus and Engelen, 2001; Eichelberger and Gudenberg, 2003; Gil and Kent, 1998; Klaus and Engelen, 2001; Dwyer, 2001).

TWO DIMENSIONAL GRAPH DRAWING

Extending the sugiyama algorithm for drawing UML class diagrams: They presented a technique for automatic layout of UML class diagrams, which is based on combination of an extension of the well known Sugiyama algorithm and orthogonal drawing techniques. They have also developed a toll called UML workbench.

Their algorithm consists of several phases, where their strategy is first: placing classes involved in inheritance relationship in hierarchical structure and

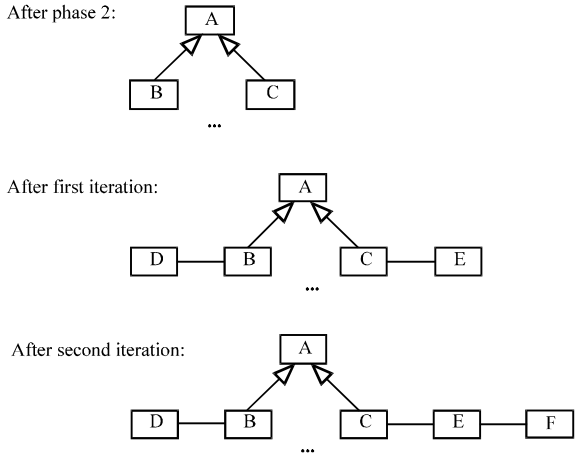

Fig. 2: Incremental extension (Seemann, 1997)

second: placing the remaining classes preserving the basic structure

Preparation: If we assume a UML class diagram as a graph G, then in this phase the direct cycles are removed. A subgraph I of G, is computed which contains only the classes related through inheritance together with their inheritance connections.

Sugiyama layout: By using Sugiyama algorithm as described by Sugiyama *et al*. (1981), the first layering is computed. This phase reorder the nodes in each layer to reduce the number of crossings.

Incremental extension: In this phase the incrementally extension of layout will lead to that all nodes are placed in the diagram as depicted in Fig. 2.

Orthogonal drawing of association connection edges: This technique is similar to the known construction of orthogonal grid drawing. The gridlines can be shared between several edges. In this phase computes the line position of the straight lines representing the inheritance relations and segment position of association relationships between classes on the same layer or on adjacent.

Graph Drawing aesthetics and the comprehension of UML class diagrams: The study was conducted by designing two experiments: the first (Experiment A) and (Experiment B). The main difference between experiment A and experiment B was the way in which the experimental diagrams were produced. While experiment A used computational metrics to determine the presence of an aesthetic in a diagram, in experiment B, a separate human perception study was used to assess the extent to which aesthetics were perceived in a diagram. Another two other important aspects which differs Experiment B from Experiment are: choice of aesthetics and aesthetic variation.

Table 1: Graph drawing aesthetics criteria

Experiment A	Experiment B
Aesthetic criteria	Edge lengths
Minimize bends	
Node distribution	Symmetry
Edge variation	
Direction of flow	
Orthogonality	

The aim of experimental was to determine which of the aesthetics underlying common graph drawing algorithms are most suited to human comprehension of UML diagrams. By asking subjects (University Students) to perform comprehension tasks on the same UML diagram portrayed with different aesthetic emphases, we aimed to identify the aesthetic criteria that resulted in the best performance.

Graph drawing aesthetics used in two experiments are shown in Table 1.

Experiment result: The results showed that none of the aesthetics really matter and therefore the human don't realize the differences between two UML support tools that use automatic layout algorithms comprising different aesthetics.

They suggested that additional semantic issues need to be considered when a layout algorithm is used in a domain-specific tool.

UML class diagrams-state of the art in layout techniques: Overview of UML Tools shows that most of them are too implementation-specific. The abstract concepts like association classes, higher association, constraints and even comments, which cannot be directly realized in a programming language, cannot be visualized. In most tools package-like elements like subsystems or models are not present.

A class diagram can be described in terms of graph theory. It is obvious that classes, packages and rhombs representing n-ary associations map to nodes. Edges are mapping of the associations, dependencies as well as inheritance relations.

Unfortunately most layout algorithms on class diagrams do not adhere to any aesthetic principles. The different survey shows that most of them don't consider the underlying semantics and further rely on aesthetical principle taken from graph drawing.

Eichelberger (2002a, b) in his previous papers discussed semantic based aesthetic criteria in order to

Table 2: Rules for drawing class diagram based on aesthetic criteria

No.	Rules
1	Enforce hierarchy as the most appropriate ordering criterion for edges in a class diagram
2	Respect spatial relationships to encode coupling, cohesion and importance of parts of the diagram
3	Visualize the natural clustering of nodes according to semantically reasons like containment, n-ary associations and patterns
4	Avoid crossings and overlapping of model elements
5	Center position of selected nodes (n-ary associations, pattern nodes)
6	Respect the vicinity of association classes, notes and constraints
7	Clearly assign adornments to edges and reflective associations to the connected classes
8	With the minimum priority respect other graph drawing criteria

Table 3: A subset of aesthetic principles

No.	Aesthetic principles
1	Identify a pseudo hierarchy by heuristics or by respecting a user defined hierarchy
2	Perform a semantic ordering to release implicit dependencies between the sequence of definitions of the model elements in the input and the layout result
3	Insert containment relations of model elements as hierarchical edges
4	Compress association classes and their edge connector nodes into compound nodes
5	Convert annotations and connected model elements to compound nodes
6	Remove reflexive associations in order to simplify the implementation. Represent the edge information within the connected classes in order to be drawn as edges later on
7	Transform the graph to an acyclic graph
8	Guarantee a virtual root
9	Calculate the ranking of the hierarchically connected nodes in one step, calculate the layer positions of only non-hierarchically connected nodes in a second step. Optimize the layered structure of the graph for UML class diagram layout
10	Calculate edge crossing minimization on hierarchical and non hierarchical edges by an incremental crossing reduction approach. Respect cluster and containment relations in this step
11	Remove containment information
12	Calculate the coordinates of nodes and edges. Contained model elements are treated in the same step in order to respect non-hierarchical edges
13	Expand compound nodes for association classes
14	Expand and layout notes

provide intuitive rules for drawing class diagram which are shown in Table 2. These rules lead to readable diagrams and therefore can reduce the cost of communication when interchanging software development diagrams. But validating these rules by user experiments is a hard task.

The Algorithms in many CASE tools mainly focus on classes, inheritance relations and association relations but not on nested package and class structures and more sophisticated model elements like association classes, higher associations or constraints.

Table 3 shows a subset of proposal for aesthetic principles for class diagrams. As a consequence of applying these principles the readability and understandability is enhanced.

Their proposed algorithm is capable of working on nested and structured elements and it can easily be updated. Although, adding more constraints to an

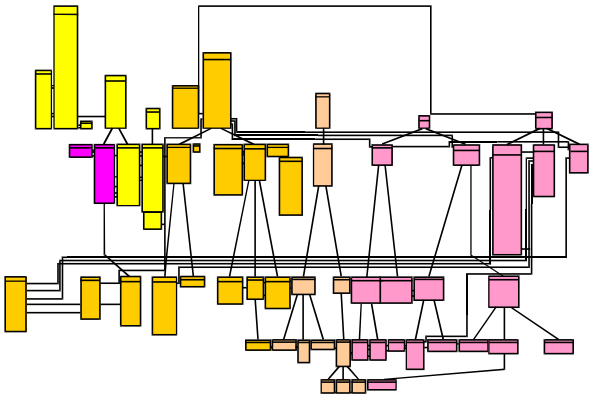

Fig. 3: A class hierarchy, where the nodes are colored and grouped according to package containment without showing the packages it selves (Eichelberger and Gudenberg, 2003)

algorithm can increase runtime and decrease quality, but introducing additional criteria into the new edge crossing reduction algorithm can be respected. They believe that additional minor update to the set of aesthetic principles is necessary. Figure 3 depict one of the result produced by their algorithm.

3D DIMENSIONAL GRAPH DRAWING

Three dimensional software modeling: They have used the third dimension to demonstrate a different kind of arrow in the same diagram. This has been presented by a series of 3D notations, which have a number of advantages over their 2D matching parts. The notations complement and are compatible with existing 2D notations such as those making up UML. In other word, it can thought of as a seamless combination of several 2D diagrams. Software models, which support 3D, enable visualization of richer semantic then those of 2D models.

The 3D notations they developed are: contract boxes, 3D sequence diagrams, nested and structured box diagram.

The Contract boxes, as depicted in Fig. 4, visualize behavior where in UML can only be shown using a combination of state diagrams and precise textual language.

The 3D sequence diagram provides much richer visualizations where it is only possible with UML sequence and collaboration diagram. The 3D sequence diagram can also show message passing between objects and sets of objects.

The nested box diagram is a combination of the contract box specifications with sequence diagram. It

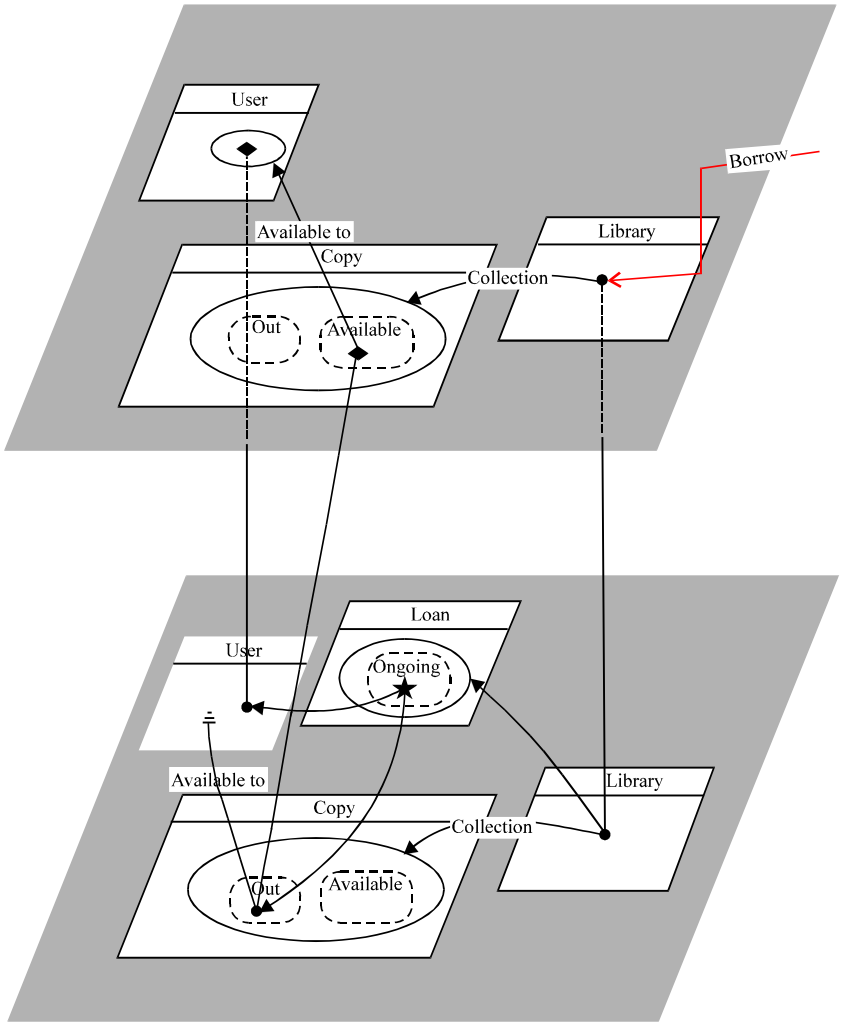

Fig. 4: Contract diagram (Dwyer, 2001)

visualizes the complete action design which includes the specifications of the sub-actions involved.

The structured box diagrams are an extension of nested box diagrams to visualize looping and conditional behavior. Hence structured box diagrams show all aspects of the model on a single diagram. All other diagrams may be considered as either sub-diagrams or projections of these.

Experience in 3-dimensional visualization of java class relations: Syntax graph is a formal model considering the structure of a system which UML visualizes some parts of the syntax graph in 2D dimensions. Their approach is a combination of visualizing the syntax graph and the benefit of visualizing the large software systems structures in a 3D dimension view.

They concentrated on the differences between 2D and 3D, so they selected out some aspects in 3D, which is much weaker in 2D. Those aspects were: transparency of objects, effect of depth, ordering in space and motion.

For geometric arrangement in 3D they used: Cone Tree, as shown in Fig. 5, where the rotation of each cone in the tree allows focusing on certain information without losing its context.

Information cube, as shown in Fig. 6, to glue together related information and Information Landscape, which uses metaphor of a landscape as shown in Fig. 7.

To show at the same time all relationships in given Java software using one of the above mentioned techniques is impossible. Thus their approach for visualizing Java software, which is statically structured by packages, interfaces and classes, in 3D space strongly,

focuses on concurrent display of different kinds of relationship by using different arrangement techniques in one diagram (combination of techniques).

Their graph-like approach is shown in Fig. 8, where nodes in the graph are classes represented by boxes, interfaces are represented by spheres and relations represented by pipes. The package membership of these classes is presented by an information cube and class hierarchy is shown as a cone tree. In the background, additional packages are shown to illustrate relationships between these packages.

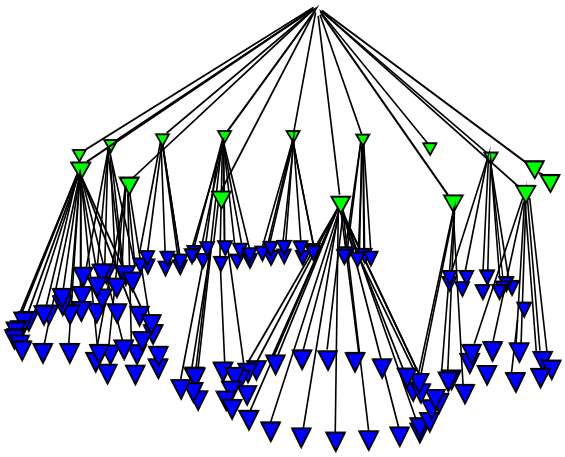

Fig. 5: A cone tree (Dwyer, 2001)

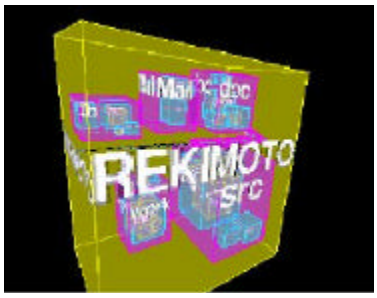

Fig. 6: A famous cube information (Dwyer, 2001)

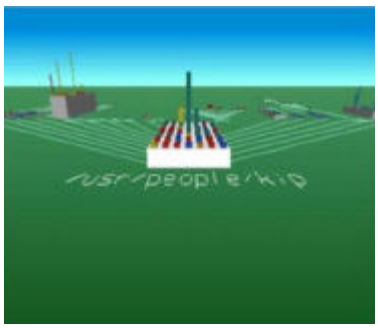

Fig. 7: Information landscape (Dwyer, 2001)

As visualization in three dimensions demands tool support, they developed a visualization tool called J3Browser. J3Browser consists of two parts: the first part analyzes Java source code and generates a structure model. The second part is a Java applet, which interactively allows applying visualization techniques on the structure model and thereby creates a 3D model represented as a VRML scene.

Three dimensional UML using force directed layout: Representations of connected graphs through 3D have a number of advantages comparing to 2D, in order to increase user's perception. Most of the tools manipulating UML models present it only in the form of 2D diagrams. In this study, they considered primarily the Class and Object diagrams from the Design View in order to examine how these diagrams can be better understood through 3D visualization.

Force Directed Algorithms (FDA), which was introduced into graph drawing 1984, is as effective as any other popular layout algorithm. This algorithm uses heuristics based on balancing a set of simulated physical forces between the elements of the graph to find optimal layout. In order to visualizing 3D UML FDAs scale up to 3D space by replacing 2D vector arithmetic with 3D vector

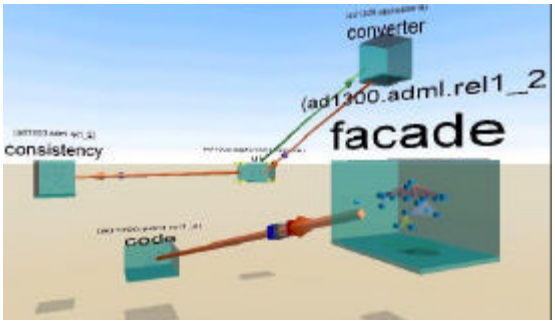

Fig. 8: Exemplifying a software structure (Dwyer, 2001)

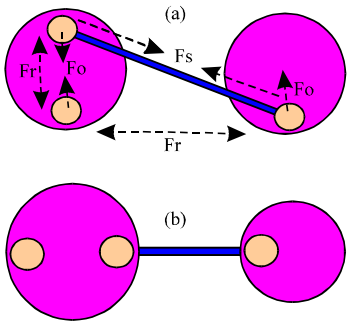

Fig. 9: Fr is Repulsion, Fs is Spring and F0 is Origin forces (Dwyer, 2001). (a) initial layout and (b) balanced layout

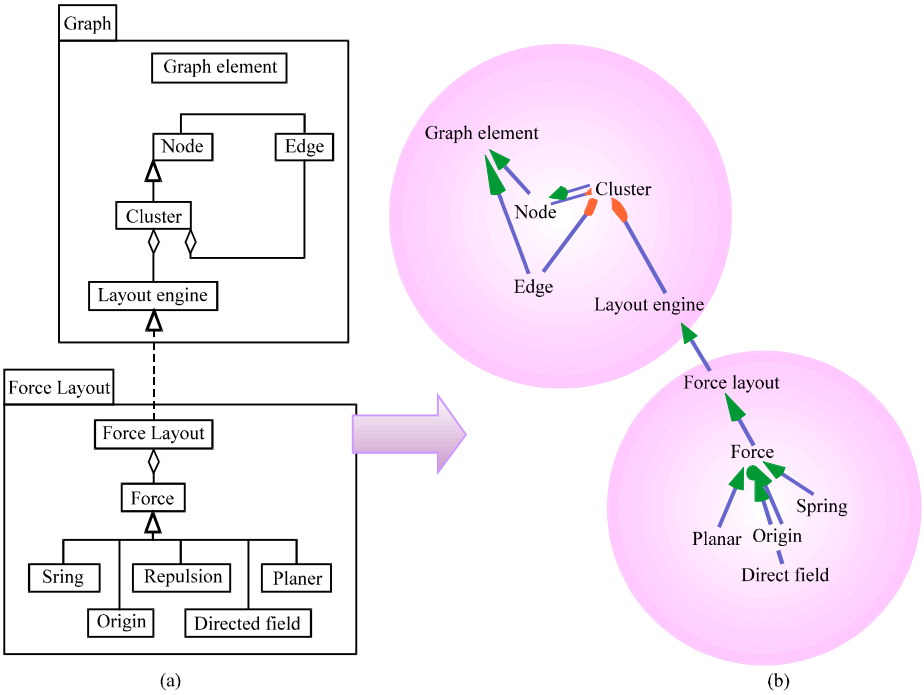

Fig. 10: (a) A Class diagram showing the essential classes implementing Wilma's graph and force model engines and (b) A 3D interpretation of the UML model in (a) showing packages as clusters (Dwyer, 2001)

arithmetic. FDAs also simulate forces (Repulsion, Spring and Origin) found in nature. Figure 9 showing all forces reaching a balanced state.

Figure 10b is a 3D interpretation of the class diagram given in Fig. 10a, where some impact of the visualization will be lost when it is printed out.

To test the success of their paradigm they did an empirical study by interviewing experienced system architects. In general the subjects answered very well to the questions about the models shown to them. One of the interesting comments shows the successful of their study:

Being able to see the diagrams from different angles allows you to emphasis different parts of the structure and really lets you see the tree in your head.

DEVELOPED TOOLS AND THEIR DOMAIN OF USE

The task of analyzing your system, or marketing challenges and defining systems and applications that can effectively address these challenges is an extraordinarily complex undertaking. There are many developed software tools that helps you control this complexity by helping you understand the problem space, define and communicate effective solutions and manage change throughout the life cycle. With the tools and techniques you are enabled to solve the right problem and define the right solution. Table 4 present some of these tools with respect to their domain of use.

Table 4: Domain of use

Domain of use	Developed tool
Interactive application as CASE tools	
Reveres-engineering tool for C++	UML workbench
Visualizing large system's structure	J3Browser
Visualizing UML diagrams, software modeling	Wilma
Visualizing UML diagrams	SugiBib
UML Class diagrams	TNI opentool
Visualizing UML diagrams	NoMagic MagicDrawUML
Modeling suite aids in deployment of Standards across corporate Development projects	Popkin system architect
Modeling suite aids in system developing from requirements	Rational analyststudio

SUMMARY

A picture is worth a thousand words is an old Chinese proverb, thus visualizing system's architecture will increase the human's perception of that system. In this paper we reviewed some papers regarding the 2D and 3D drawing algorithms from a technical and strategy view.

Although, representations of connected graphs through 3D have a number of advantages comparing to 2D, but some impact of the visualization will be lost when

it is printed out. 3D representation is really proposed for interactive use in an animated, navigable, color graphics environment. Our future work is to investigate the capability of these algorithms to visualizing of interactions between components composed by CIML.

We also showed in which area or domain, as shown in Table 4 they are mostly used. This review shows that there is potential for research in this area.

REFERENCES

Dwyer, T., 2001. Three dimensional UML using force directed layout. Proc. Aust. Symp. Inform. Visualisat., 9: 77-85.

Eichelberger, H. and J.W. Gudenberg, 2003. UML class diagrams-state of the art in layout techniques. Proceedings of the Visualizing Software for Understanding and Analysis (VISSOFT'03), Sept. 22, Amsterdam, Netherlands, pp: 30-34.

Eichelberger, H., 2002a. Aesthetics of class diagrams. Proceedings of the 1st IEEE International Workshop on Visualizing Software for Understanding and Analysis, June 26, IEEE, Computer Society Washington, DC, USA., pp: 23-31.

Eichelberger, H., 2002b. Evaluation-report on the layout facilities of UML tools. Technical Report 298, Institut fur Informatik, University of Wurzburg, July 2002.

Gil, J. and S. Kent, 1998. Three dimensional software modeling. Proceedings of the 20th International Conference on Software Engineering, April 19-25, IEEE Computer Society Washington, DC, USA., pp: 105-114.

Klaus, A. and F. Engelen, 2001. Experiences in 3-dimensional visualization of java class relations. J. Integ. Des. Process Sci., 5: 91-106.

OMG., 2001. OMG unified modeling language specification. Version 1.4, February 2001.

Purchase, H., M. McGill, L. Colpoys and D. Carrington, 2001. Graph drawing aesthetics and the comprehension of UML class diagrams: An empirical study. Proceedings of the Asia-Pacific Symposium on Information Visualisation, Dec. 01, Sydney, Australia, pp: 129-137.

Sanatnama, H., A.A.A. Ghani, R. Atan and M.H. Selamat, 2009. Components interaction markup language for mediator connector. J. Applied Sci., 9: 1046-1055.

Seemann, J., 1997. Extending the sugiyama algorithm for drawing uml class diagrams: Towards automatic layout of object-oriented software diagrams. Proceedings of the 5th International Symposium on Graph Drawing, Sept. 18-20, Addison-Wesley, pp: 415-424.

Sugiyama, K., S. Tagawa and M. Toda, 1981. Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybernet., 11: 109-125.