Software Model Creation with Multidimensional UML

Lukáš Gregorovič[1], Ivan Polasek[1(✉)], and Branislav Sobota[2]

[1] Faculty of Informatics and Information Technologies Institute of Informatics and Software Engineering, Slovak University of Technology in Bratislava, Bratislava, Slovakia
{xgregorovic,ivan.polasek}@stuba.sk
[2] Faculty of Electrical Engineering and Informatics, Department of Computers and Informatics, Technical University of Košice, Košice, Slovakia
branislav.sobota@tuke.sk

Abstract. The aim of the paper is to present the advantages of the Use Cases transformation to the object layers and their visualization in 3D space to reduce complexity. Our work moves selected UML diagram from two-dimensional to multidimensional space for better visualization and readability of the structure or behaviour.

Our general scope is to exploit layers for particular components or modules, time and author versions, particular object types (GUI, Business services, DB services, abstract domain classes, role and scenario classes), patterns and anti-patterns in the structure, aspects in the particular layers for solving cross-cutting concerns and anti-patterns, alternative and parallel scenarios, pessimistic, optimistic and daily use scenarios.

We successfully apply force directed algorithm to create more convenient automated class diagrams layout. In addition to this algorithm, we introduced semantics by adding weight factor in force calculation process.

Keywords: 3D UML · Analysis and design · Sequence diagram · Class diagram · Fruchterman-Reingold

1 Introduction

Increasing requirements and the complexity of designed systems need improvements in visualization for better understanding of created models, for better collaboration of designers and their teams in various departments and divisions, countries and time zones in their cooperation creating models and whole applications together.

In software development, Unified Modeling Language (UML) is standardized and widely used for creation of software models describing architecture and functionality of created system [4].

There are many tools that allow creation of UML diagrams in 2D space. Moving UML diagrams from two-dimensional to three-dimensional space reduces complexity and allows visualization of the large diagrams in modern three-dimensional

I. Khalil et al. (Eds.): ICT-EurAsia 2015 and CONFENIS 2015, LNCS 9357, pp. 343–352, 2015.
DOI: 10.1007/978-3-319-24315-3_35

graphics to utilize benefits of the third dimension and achieves more readable schemas of complex models to decompose structure to particular components, type layers, time and author versions.

We need to decompose behaviour and functionality to particular scenarios of the system, alternative and parallel flows, pessimistic, optimistic and daily use scenarios.

2 Related Work for 3D UML

There are some existing alternatives how to visualize UML diagrams in 3D space. Paul McIntosh studied benefits of the 3D solution compared to traditional approaches in UML diagrams visualization. Because of using combination of X3D (eXtensible 3D) standard and UML diagrams, he named his solution X3D-UML [9]. X3D-UML displays state diagrams in movable hierarchical layers [3].

GEF3D [5] is a 3D framework based on Eclipse GEF (Graphical editing framework) developed as Eclipse plugin. Using this framework, existing GEF-based 2D editors can be easily embedded into 3D editors. GEF3D applications are often called multi-editor. Main approach of this framework is to use third dimension for visualization connections between two-dimensional diagrams.

Another concept in field of 3D UML visualization is on virtual boxes [8]. Authors placed diagrams onto sides of box allowing them to arrange inter-model connections which are easily understandable by the other people. GEF3D does not allow users to make modifications in displayed models. Due to fact that UML diagrams can be complex and difficult to understand, geon diagrams [7] use different geometric primitives (geons) for elements and relationships for better understanding [11].

3 Our Approach

Our method visualizes use case scenarios using UML sequence diagrams in separate layers all at once in 3D space, transforms them to the object diagrams (again in separate layers) and automatically create class diagram from these multiple object structures with real associations between classes to complete structure of designed software application.

Sequence diagrams in 3D space of our prototype allow to analyse and study process and complexity of the behaviour simultaneously and compare alternative or parallel Use Case flows.

Identical elements in object diagrams have fixed positions for easy visual projection to the automatically created class diagrams with classes derived from these objects. Their relationships (associations) are inferred from the interactions in the sequence diagrams and class methods are extracted from the required operation in the interactions of these sequence diagrams.

3.1 Our Prototype

We have created our prototype as a standalone system in C++ language with Open Source 3D Graphics Engine (OGRE) or OpenSceneGraph as an open source 3D

graphics application programming interface and high performance 3D graphics toolkit for visual simulation, virtual reality, scientific visualization, and modeling.

For integrated development environment (IDE) we can use Eclipse or Microsoft Visual Studio and build standalone system with import/export possibilities using XMI format (XML Metadata Interchange) or plugin module to IBM Rational Software Architect or Enterprise Architect.

Our prototype allows to distribute diagrams in separate layers arranged in 3D space [10]. In this tool is possible to create UML class diagram, sequence diagram and activity diagram in multidimensional space with 3D fragments [6].

Layers can be interconnected and diagrams can be distributed to the parts in these separate layers to study interconnections and for better readability.

3.2 Diagram Transformation

In software analysis and development is good practise to start with describing and capturing system behaviour. For this purpose of behavioural modeling we can use sequence diagrams.

```
classReferences = {};
foreach layer in layers do
    foreach lifeline in lifelines do
        sourceClassName ← getClassName(lifeline);
        if sourceClassName not in classReferences then
            append Class(sourceClassName);
        end
        foreach message in lifeline do
            source ← sourceLifeline(message);
            target ← targetLifeline(message);
            targetClassName ← getClassName(target);
            if targetClassName not in classReferences then
                append Class(targetClassName);
            end
            appendMethod(targetClass, message);
            createAssociation(sourceClass, targetClass);
        end
    end
end
```

Algorithm 1. Class diagram creation algorithm

Creating sequence diagrams we automatically identify essential objects and their methods that are necessary for functionality of the system. Thanks to element similarities between sequence diagram and object diagram in the UML metamodel definition, it is possible to use same shared data representation. Object diagram can be rendered from sequence diagram. Modifications are made in drawing algorithms. Instead of drawing full timeline graphic, lifelines are ignored and only upper part with object names is drawn. Messages between lifelines are moved from original position to directly connect appropriate objects. Transformation can be visible in Figs. 1 and 2.

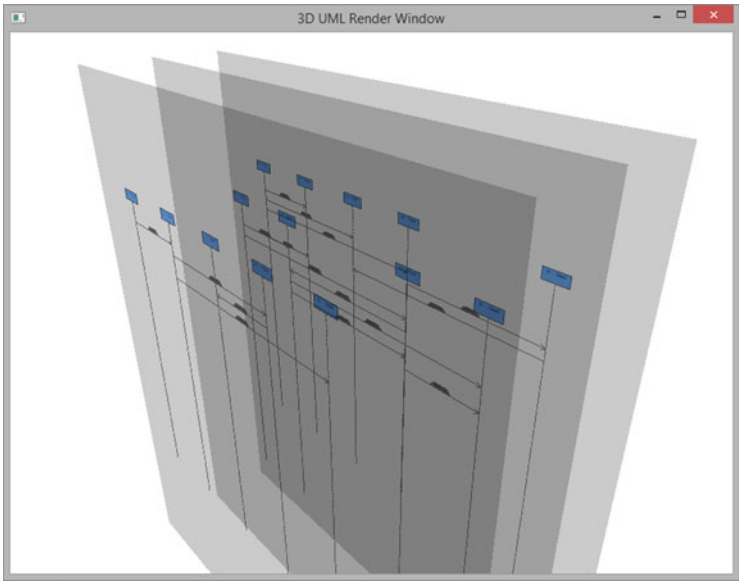

Fig. 1. Example of sequence diagrams in 3D UML.

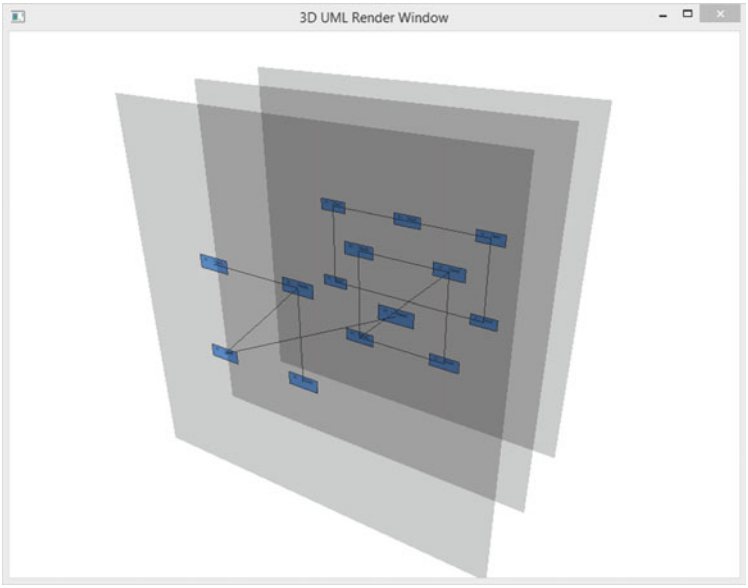

Fig. 2. Object diagrams rendered from sequence diagram.

For development in early phases of testing the concept we used layout algorithm that can be seen in Algorithm 1. Each unique element was placed on next available cell in imaginary grid. Advanced layout creation with force-directed algorithms is described in the next section of this paper.

Class diagram is gradually created. Instead of multiple passes through sequential diagrams to create the classes and append methods to these classes in the next iteration, algorithm for class diagram creation was optimised with buffering and memorisation. Each time when unknown class type is found for lifeline, new class instance in class diagram is created and reference is stored under unique identifier matching class name. Class types are then complemented with method names.

4 Class Diagram Layout

In transformation process we use some basic grid algorithms to arrange the objects in the matrix. With the growing number of the objects also grows the complexity of the diagram and relations between vertices, so it is crucial to create layout that is clear and readable.

4.1 Force-Directed Algorithms

One way how to accomplish better layout is to use force-directed algorithms, so the diagram will be evenly spread on the layer and elements with real relations are closer to each other as to the other elements. We have tested Fruchterman-Reingold and FM3 algorithms.

Fruchterman-Reingold. Fruchterman-Reingold (FR) is simple force-directed algorithm. Each vertex is repelled from the other vertices. Edges between vertices acts as springs and pulls vertices to each other, counteracting repulsive forces.

Algorithm iterates through the graph many times and each time decreases the magnitude of changes in positions, this effect is called cooling down. It could settle in some configuration to ensure the layout instead of oscillating in some other cases. Speed of layout generating is $O(n^3)$, where n is number of vertices [1].

Fm3. FM3 algorithm is more complex approach. Basic idea of the forces is the same, but FM3 uses principle of multiple levels of layout. Main difference is in the step, where provided graph is reduced into smaller subgraphs by packing multiple vertices into one. Analogy of this principle is based on finding so-called *solar systems*, where one vertex is identified as the sun and the other edges that are related to the sun are marked as the *planets* and the *moons*.

Reduction of the graph is recursively called on subgraphs until simple graph is reached, then the subgraphs are arranged and unfold, so it is returned to its higher graphs (see Figs. 3 and 4). These steps are repeated until we reach original graph. Last step arranges the final graph.

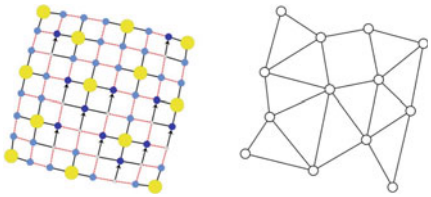

Fig. 3. FM3 - solar systems collapsed into subgraph [2]

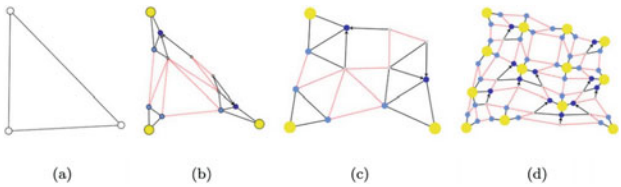

Fig. 4. FM3 - unfolding sub-graphs and layout creation [2]

This solution is significant quicker than Fruchterman-Reingold algorithm. It is possible to reach speed O(|V|log|V|+|E|) [2].

4.2 Problems of Force-Directed Algorithms

Unfortunately the outputs of these algorithms were not good enough for proposed use. More appropriate applications are the visualisation of large graph with tree structures.

Both these algorithms have a tendency to create uniform distribution of elements in diagram. Users have a tendency to arrange elements into groups, order elements by priority, hierarchy and so on. They are looking for patterns, relations, semantics and other hidden aspects of model. This is important factor for conservation the readability and understandability of the modelled diagrams. Deficiency of this features in force directed algorithms make them not ideal to create class diagram layout.

Our focus in this phase was on the creation of the algorithm that is more appropriate. Starting point and proof of concept was considering simple semantics in diagram layout creation. Assuming that in class diagram the most important relation between two elements from semantic view is generalisation, then aggregation and finally association, it is possible to modify output of layout algorithm by adding weight factor in attractive force calculation process.

Analysing mainly two mentioned force-directed algorithms (but also the others) was created some methods how to accomplish the task of incorporation the semantic into the selected algorithms.

In case of Fruchterman-Reingold it is possible to introduce weight to vertices or edges. By adding weight, it is possible to modify the original behaviour of algorithm.

Modifying process of solar systems selection in FM3 could allow to create sub-graphs, where semantically relevant objects are merged into one vertex. This ensures the separation of less relevant parts of diagram in space and then the layout is enriched

by adding more elements in relevant places by reversing graph into its higher sub-graphs.

Our decision was to utilise Fruchterman-Reingold algorithm. Time complexity of the algorithm in comparison with FM3 does not become evident according to the scale in which we use these algorithms: class diagram with size of 10-100 classes, layout calculation is fast. Implementation of the algorithm is simple and it is possible to make modifications more easily than in FM3. Implementation using FM3 can be realized in the future if it.

4.3 Weighted Fruchterman-Reingold

Simple modification of FR algorithm in the form of adding weight to edges in calculation of attractive forces make desired layout improvement. Weight of edge is taken into account in process of calculating attractive forces of edge connecting two vertices.

Calculated force is multiplied by weight of corresponding type of edge. It is necessary to identify current type of the edge while calculating attractive forces. Implementation of the system distinguishes different relations as instances of different classes and therefore it is easy to use appropriate weight.

While prototyping phase, weights of relations-edges were experimentally set as follows:

- generalisation → 200
- aggregation → 100
- association → 10

Application of the selected weights affected the outputs of the algorithm in desired manner. To escalate effects of attraction, reflecting the semantics of the diagram, vertices are repelling each other with equivalent force, but magnified by factor of 10 according to original force calculated by algorithm. This tends to push vertices more apart, so the difference in distances between related and unrelated vertices is greater. This allows to make semantics patterns of class diagram more visible.

5 Results and Evaluation

New weighted Fruchtermant-Reingold algorithm was tested against Fruchterman-Reingold algorithm. Empirical comparison of generated layouts on multiple class diagram examples indicates, that our new algorithm provides more appropriate layout.

First example in Figs. 5 and 6 shows one of tested class diagrams: Sequence diagram metamodel. Differences between both layouts are clearly visible. Figure 6 shows layout generated by Fruchterman-Reingold algorithm. This layout is evenly distributed across available space and it has symmetrical character.

Distribution of the classes is not optimal and orientation in such diagrams is still not easy and natural. Random scattering of connected classes is not very useful in case of readability and understanding of created class diagram.

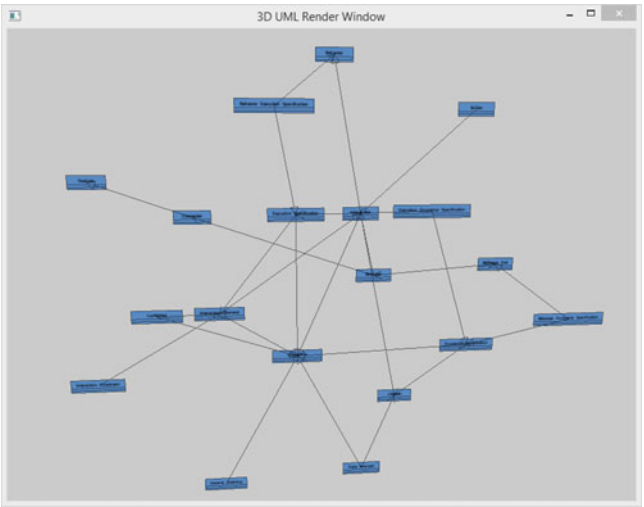

Fig. 5. Sample (Sequence diagram metamodel) - layout generated with Fruchterman-Reingold algorithm

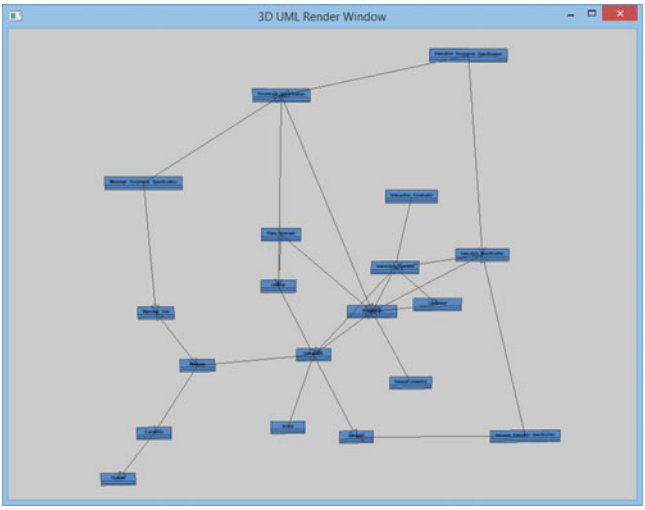

Fig. 6. Sample (Sequence diagram metamodel) - layout generated with Weighted Fruchterman-Reingold

Using weighted Fruchterman-Reingold algorithm on the same class diagram example achieves significantly better layout. Main difference is that layout put relevant classes more together and creates smaller chunks of classes instead of one big mass as it was in the previous case. This means better readability, understanding and modifying designed diagrams.

Algorithm still creates some unwanted artefacts. For example, by pushing on some classes creates unnecessary edge crossings. These problems may be addressed in the future.

Nevertheless algorithm is able to create decent layout for the class diagram. Output is not perfect, but it is an initial layout, which could be corrected by the user: weighted Fruchterman-Reingold algorithm is suitable for this purpose.

6 Conclusion

We applied force directed algorithm successfully in the second phase of transformation from object diagrams (derived from use case scenarios in sequence diagrams) to class diagram representing static structure of the modeled software system.

In addition we introduced semantics by adding weight factor in force calculation process in the layout algorithm. Type of relation between vertices influence weight applied on the attractive forces. This creates more useful layout organisation, as elements are grouped by semantics that is more readable.

Research started with software development monitoring [13] and software visualization [12] and now, we are preparing interfaces and libraries for leap motion, 3D Mouse, and Kinect to allow gestures and finger language for alternative way of creating and management of the particular models.

Acknowledgement. This work was supported by the KEGA grant no. 083TUKE-4/2015 "Virtual-reality technologies in the process of handicapped persons education" and by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/1221/12.

This contribution is also a partial result of the Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

References

1. Fruchterman, T.M., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Experience **21**(11), 1129–1164 (1991)
2. Hachul, S., Jünger, M.: Large-graph layout with the fast multipole multilevel method. Online verfügbar unter http://www.zaik.uni-koeln.de/∼paper/preprints.html (2005)
3. McIntosh, P., Hamilton, M., van Schyndel, R.: X3d-uml: Enabling advanced uml visualisation through x3d. In: Proceedings of the Tenth International Conference on 3D Web Technology, Web3D 2005, pp. 135–142, New York, NY, USA, 2005. ACM
4. OMG. OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1, August 2011
5. Pilgrim, J., Duske, K.: Gef3d: a framework for two-, two-and-a-half-, and three-dimensional graphical editors. In: Proceedings of the 4th ACM Symposium on Software Visualization, SoftVis 2008, pp. 95–104, New York, NY, USA, 2008. ACM
6. Škoda, M.: Three-dimensional visualization of uml diagrams. Diploma project, Slovak University of Technology Bratislava, Faculty of informatics and information technologies, May 2014

7. Casey, K., Exton, Ch.: A Java 3D implementation of a geon based visualisation tool for UML. In: Proceedings of the 2nd international conference on Principles and practice of programming in Java, Kilkenny City, Ireland, 16–18 June (2003)
8. Duske, K.: A Graphical Editor for the GMF Mapping Model (2010). http://gef3d.blogspot.sk/2010/01/graphical-editor-for-gmf-mapping-model.html
9. McIntosh, P.: X3D-UML: user-centred design. implementation and evaluation of 3D UML using X3D. Ph.D. thesis, RMIT University (2009)
10. Polášek, I.: 3D model for object structure design (In Slovak). Systémová integrace **11**(2), 82–89 (2004). ISSN 1210–9479
11. Ullman, S.: Aligning pictorial descriptions: an approach to object recognition. Cognition **32**, 193–254 (1989)
12. Polášek, I., Uhlár, M.: Extracting, identifying and visualisation of the content, users and authors in software projects. In: Gavrilova, M.L., Tan, C., Abraham, A. (eds.) Transactions on Computational Science XXI. LNCS, vol. 8160, pp. 269–295. Springer, Heidelberg (2013)
13. Bieliková, M., Polášek, I., Barla, M., Kuric, E., Rástočný, K., Tvarožek, J., Lacko, P.: Platform independent software development monitoring: design of an architecture. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 126–137. Springer, Heidelberg (2014)