Scientific Journal of Riga Technical University
Computer Science. Applied Computer Systems
2011
Volume 47

UML Diagram Layouting: the State of the Art

Arthur Galapov[1], Oksana Nikiforova[2], [1-2]*Riga Technical University*

***Abstract* – The usual aim of the modern computer-aided system modelling is to improve a connection between software model and code components. Therefore, the task of a diagram import/export becomes very important during software development. Layouting of diagrams after importation from another tool and application plays the main role. Authors of this paper describe some concepts, which are currently being considered in the area of diagram layouting and indicate several problems and their potential solutions for use in the development of CASE tools.**

Keywords – Diagram layouting, element placement, model export/import, system model.

1. INTRODUCTION

One of the tasks of software development is to present different aspects of the system before developing software solution for the required system. Solving this task, system modelling becomes one of the important activities during software development.

System modelling gives software developers the ability to understand system's behaviour, structure and its separate elements. System modelling is a way of thinking about problems using models, which are based on real-world ideas. Models are useful for understanding problems, communicating with everyone involved within the project (customers, domain experts, analysts, designers etc), modelling enterprises, preparing documentation and designing applications and databases. Modelling promotes better understanding of requirements, clearer view of design and more maintainable systems.

Usually, system model is organized as a set of diagrams, where specific notation is defined for each diagram and it regulates diagram syntax and semantic. As far as system models are abstractions that display the essentials of a complex problem or structure by filtering out nonessential details, models make the problem easy to understand. Systematic approach to elements placement within the diagram, which is specified as a task of diagram layouting, plays an important role in completing the task of modelling. Increased interest in Model Driven Software Development again turns focus to the area of diagram layouting, which concords to the area of graph theory.

In 1985 several works were done on ER (Entity-Relationship) diagrams: Batini, Furlani and Nardelly in [1] described some aesthetics and applied topology- shape-metrics approach. According to Dalj, aesthetics is the theory about elegancy [2]. For network diagrams Kosak, Marks and Shieber in [3] specify two algorithms respecting certain visual organization features. The first algorithm selects and applies a layout rule until each node is positioned. The second one is a parallel genetic algorithm. Freivalds and Kikusts in [4] along with Dogrusoz in [5] propose new approaches and techniques for graph layouting.

Several researches have been conducted on layouts of class diagrams. Early work of Battista and his colleagues explored graph drawing algorithms and aesthetics [6]. Some new approaches have been proposed for graph layout especially in the UML class diagram domain. Eiglsperger, Kaufmann M and Siebenhaller in [7] proposed an algorithm based on the topology-shape-metrics approach for automatic layout of class diagrams, which works well for class diagrams with well defined relationships between classes. Eichelberger introduced a layout algorithm according to a large number of aesthetic criteria of UML class diagrams [8]. Dwyer presented a three-dimensional UML class diagram representation using the Force Directed algorithm [9]. Andriyevska and her colleges give ideas on positive aspects of layouting in [10].

Researches also have been carried out on other types of UML diagrams. Eichelberger in [11] presents research on automatic layout of UML (Unified Modelling Language) using case diagrams. Bist with MacKinnon and Murphy propose an approach to draw sequence diagrams in technical documentation to ease communication between project members [12]. Poranen with colleagues proposes various criteria for drawing a sequence diagram based on traditional graph drawing aesthetics and the special nature of sequence diagrams [13]. Wong and Dabo give the requirements set based on cognitive science for sequence and class diagrams, which can help in diagram readability improvement [14].

There are many criteria introductions that conflict with each other. It confuses software engineers and tool developers while choosing proper criteria to use. Therefore, we can suggest that the area is not systematized well enough and the goal of the paper is to summarize existing information connected with diagram elements layouting and to give systemised view on existing problems and their potential solutions, and propose more specified field for further research.

The paper is structured as follows: the next section describes general terms and definitions of the area of diagram layouting and shows classification of diagrams suggested by the authors, which can be used for working out the algorithms of element placement. The third section contains theories, which help to understand how a human being perceives objects from real world and joins them into a system or distinguishes them from background. These theories give opportunity to distinguish how to organize UML diagram elements for improving its readability and introduce the requirement set. The subsections of section 4 describe problems in several areas of working with models and how layouting automation helps in solving them. In conclusion of

the paper the authors discuss the present research and state the directions for the future.

2. Classification of Different Diagram Types on the Example of UML

As the authors have mentioned in the introduction, the task of element placement during system modelling has an impact on better understanding of the system model and more effective usage of them during development of the system. Nowadays, object oriented manner of software development plays one of the leading roles in system development and object oriented system modelling has its own way of presentation of different aspects of the system. Therefore, the problem of diagram layouting is described on the example of UML [15], which is declared as a standard for presentation of software system model. It also provides a notation, which grows from analysis through design into implementation in object oriented programming languages.

As a notation of system modelling for different aspects of the system, UML introduces different types of diagrams, which can describe a system from different points of view. According to [15] UML 2.x version distinguishes 13 diagram types, abstract examples of them are shown in Figure 1:

1) Class diagram – describes the system structure by showing its classes with methods and attributes, and relations between these classes.
2) Components diagram – shows how system is divided into components and in what manner these components relate with each other.
3) Composite structure diagram – demonstrates classes' inner structure and collaborations that this structure make possible.
4) Deployment diagram – describes the hardware used in system implementations, the execution environments and artefacts deployed on the hardware.
5) Object diagram – shows full or partial structure of modelled system at specific time.
6) Package diagram – shows how system is divided into logical parts and how these parts are connected with each other. There is no strict difference between other diagram types and the name is chosen for simplicity – packages and package diagrams can be part of other diagrams.
7) Activity diagram – shows how certain activity is divided into different actions.
8) State machine diagram – describes the states and state transitions of the system.
9) Use case diagram – describe system's functionality in terms of actors, their goals represented as use cases and dependencies between these elements.
10) Communication diagram – shows the interactions between objects or parts in terms of sequenced messages.
11) Sequence diagram – shows how objects communicate with each other in terms of sequence of messages.
12) Interaction overview diagram – diagram type, which is similar to activity diagram with one difference: diagram activities are pictured as frames, which can contain sequence diagrams.
13) Timing diagram – specific type of diagram, where the focus is concentrated on timing constraints.

We can assume that all diagrams more or less are represented in a graph form – diagram consists of nodes, which are connected with arcs in some manner. However different diagram types can have different structure: diagram can have different type of nodes or arcs, diagram should be constructed in some special manner.

The "simplest" presentation of elements from the perspective of graph structure has deployment diagram. It has two types of elements, one of them is a node, which describes physical place of system deployment, and the other is a link between nodes. The same is within the object diagram, where, in accordance with UML notation, diagram has two types of elements – objects and links between them.

Diagram having different types of arcs or nodes must be analyzed separately from diagrams with one type of arcs and one type of nodes, because extra types of elements should be taken into consideration.

In spite of simple structure using case diagram, where actors have to be communicated with use cases by relationships, the diagram has additional conditions on actors' placement around the set of use cases, which in turn specifies the boundary of the system. State chart diagram has the same structure as use case diagram. UML communication diagram has similar structure, but a bit more complicated - objects are connected with arcs, where the arcs have complicated structure. The link is presented as a connector, which is anchored with the message that object has to pass to another object. Therefore, in addition to rules of element placement accordingly to graph structure, the distance between objects has to be considered to place the name of the message. Composite structure diagram has the same structure as communication diagram: diagram has two types of nodes, which are connected with one type of arcs and their names placed on them. These four diagram types can be grouped into the diagrams, which require specific regulations for graph nodes.

UML class diagram has one type of nodes that represent system classes and several types of arcs, which show different types of class relationships. Component diagrams have one type of nodes and several types of arcs, like class diagram, but in this case different types of arcs have the same semantics and are used to improve the readability of a diagram. These two types can be joined into the diagrams, which require specific regulations for graph arcs.

Logically package diagram consists of one type of nodes that represent packages and several types of arcs that show how packages relate with each other. But in most cases package diagrams are part of other diagram types – this adds more node types to the diagram. Activity diagram also has several types of nodes: activity, entry point and exit point; and several arc types. But arc structure is complicated: arcs can split into several flows and then join into one, also arcs can be split into alternative flows. These two types of graphs can be joined into diagrams, which require specific regulations for graph arcs and nodes.

Sequence diagram has a special structure: all the objects are allocated horizontally at the top of the diagram, except objects created during system operating, each of the objects has its life time, sequence of messages is shown by links showing its flow.

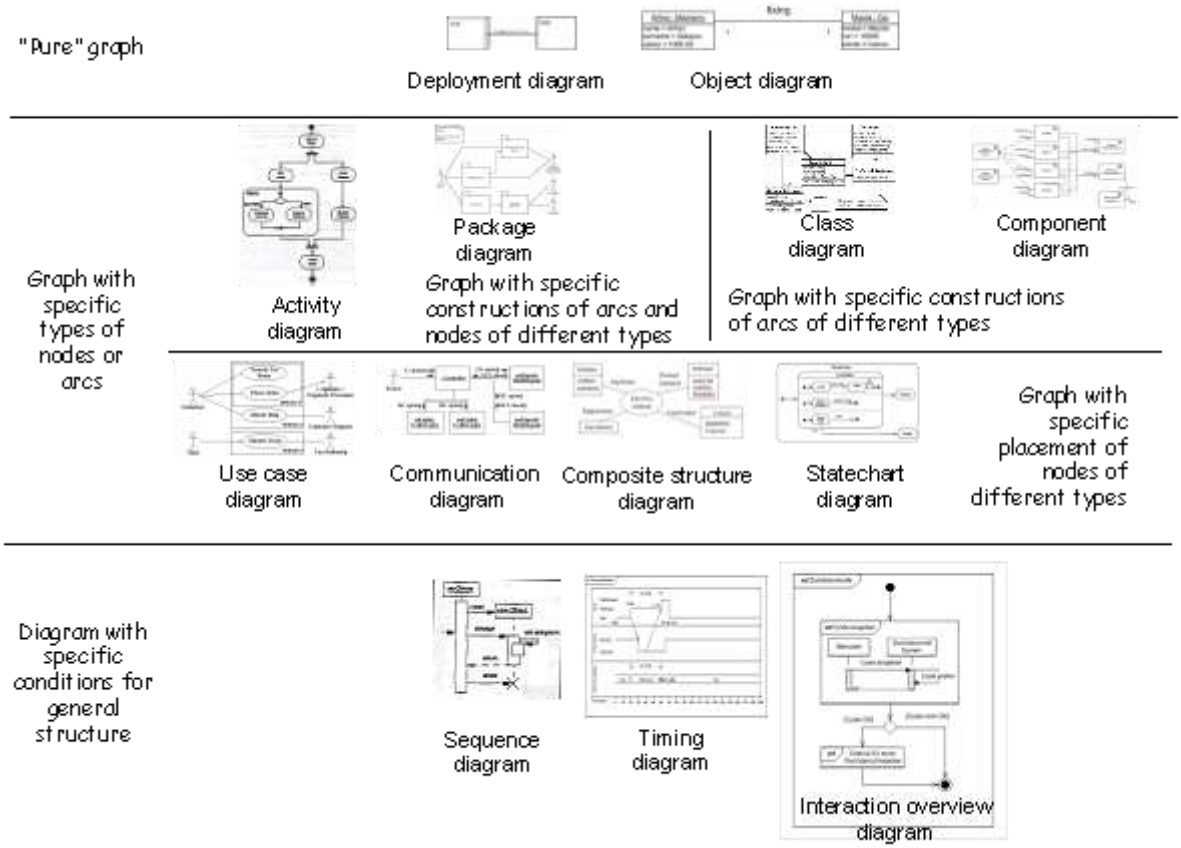

Fig.1. UML diagrams classification

Interaction overview diagram has the same structure as activity diagram, but instead of activities, nodes of interaction overview diagram can have separate sequence diagrams, also arcs have simpler structure – no flow separation and joining. Timing diagram can't be considered as a graph, because it has no nodes. These three diagram types can be joined into graphs with specific conditions for general structure.

Taking into consideration the specific requirements to structure and placement of elements of UML diagrams above, it is possible to classify diagrams in different groups, which are based on diagram structure and it construction principles. The authors propose to classify diagrams in three major groups in accordance with requirements for their layout:

1) "Pure" graph –diagrams are represented as a regular graph: one type of nodes is connected with one type of arcs.
2) Graph with nodes and arcs of different type:

a. diagrams, which are specific in correspondence with requirements to node types,

b. diagrams, which have specific requirements for the structure of arcs,

c. diagrams, which have specific requirements for the structure of nodes and arcs.

3) Diagrams with specific requirements of the diagram structure – group of diagrams, which are constructed in some special manner.

Diagrams from the second group can take a form of "pure" graph in a special case, when the diagram has only one type of nodes and only one type of arcs. This classification gives an opportunity to distinguish diagram types, which can be considered together under research of some specific methods or criteria for UML diagrams efficiency improvements.

Tilley and Huang in [16] discuss, that UML diagram efficiency depends on 4 factors: UML syntax and semantics, layout of UML diagram elements and domain knowledge. Layout is essential factor for diagram reading and comprehension, therefore, studies on diagrams aesthetics appeared, which try to explain effective diagram construction principles, and on diagram layouting algorithms, which study and develop algorithms for automation of diagrams of different type layouting.

To specify UML diagram elements layout comprehension, possible solutions for UML diagram transformation from chaos state to normal form have to be defined by analogy showing how it is made in databases designing. According to

Dalj, chaos is a state of extreme disorder and uncertainty [2]. Taking this definition in consideration, it is possible to conclude that UML diagram in the chaos state have low elements layout efficiency. According to Murdock in [17] normal form – in mathematics, object's simplified form is achieved by transformations, which don't affects this object's properties; in data bases, requirement set, which relation it must satisfy. The normal form of UML diagram is initial transformed state, which was found applying its diagram elements relocations in space and which layout satisfy requirement set (for example, perception theories). The next chapter defines the requirement set for certain UML diagram types and explains how these requirements effect UML diagram readability.

3. General layouting principles

Layout of UML diagram elements is of high importance for software system understanding. The higher the layout of a diagram elements efficiency is, the easier is its comprehension and the higher UML use efficiency is. Theory of perception is one of the bases which can help to define efficiency of layout of UML diagram elements.

To determine requirements for layouting of diagram elements there is a need to answer the question – which element layout will be more perceptible for the user? [14] explains that principles of perceptual organization and segregation provide the basic design rules to organize multiple artefacts. These rules can help users to group related information and segregate useful information easily and without ambiguity. So theory of perception can help to distinguish certain criteria for UML diagram effective layouting.

A. Perceptual organization

According to Boff, Kaufman and Thomas in [18] - perceptual organization refers to how objects that we perceive in the world are located and related with one another. Wong and Dabo defined the most important laws of object perceptual organization in [14], such as:

1) Images are perceived in a way that their structures are as simple as possible. The law of good figure is also called the law of simplicity.

2) Law of similarity – similar elements (e.g., common in shape or colour) appear to be grouped together. For example, in figure 2, (a) can be perceived as either horizontal rows or vertical columns of circles, but (b) would most likely be perceived as vertical columns of squares and circles because of the similar shapes.

3) Law of continuation – points tend to belong together if they result in straight or smoothly curved lines when connected, and lines are grouped together in such a way as to follow the smoothest path.

4) Law of proximity – elements that are close to each other are grouped together. For example, in figure 3 (a), the circles are more likely to be perceived as horizontal rows since they are closer horizontally. Also figure 3 (b) most likely will be perceived as horizontal lines of circles and squares since they are closer horizontally; disregarding the fact that similar shapes are placed vertically. Conclusion – the law of proximity is stronger than law of similarity.

5) Law of connectedness – elements that are physically connected are perceived as a unit. In figure 4, we perceive three dumbbells rather than some pairs of dots. Note that the dots that are next to each other in adjacent dumbbells are actually closer together, and according to the law of proximity, they should be grouped together. However, in this case, the law of connectedness overpowers the law of proximity.

6) Law of familiarity – elements are more likely to be grouped together if the groups seem familiar or meaningful.

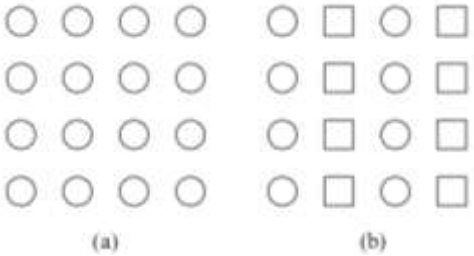

Fig.2. Examples of the law of similarity [14]

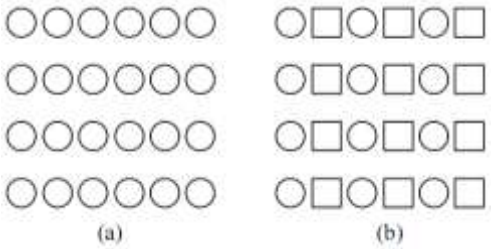

Fig.3. Examples of law of proximity [14]

Fig.4. Example of law of connectedness [14]

Perceptual organization helps to understand how human beings' perceptual system combines separate objects into system. This helps to determine how to organize UML diagram elements so these elements would be comprehended as one.

B. Perceptual segregation

As it is said in Goldstein research in [19] - compared to perceptual organization, research in perceptual segregation basically studies the problem of "figure-ground segregation" to determine when objects (figures) are seen as separate from the background (the ground). After summarizing perceptual segregation laws from Dabo and Wong [14] work and Kimchi and his colleagues [20] work it is possible to distinguish these as most important:

1) Law of symmetry – symmetric areas are usually seen as a distinct figure.

2) Law of orientation – horizontal or vertical orientations have higher probabilities to be seen as a figure than other orientations. For example, in 5th figure's (a) subpicture, the vertical/horizontal "plus" propeller shape (b) is more likely to be perceived as a figure than the tilted "cross" shape (c).

3) Law of contour – modern theories about figure-ground segregation discovered that contours (i.e., boundary edges) help in figure-ground perception.

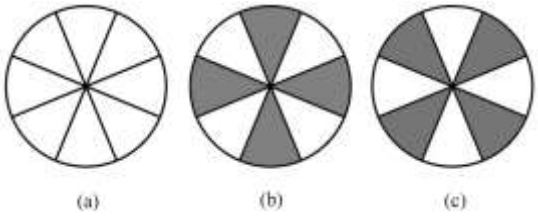

Fig.5. Example of law of orientation [14]

Perceptual segregation principles give possibility to understand how human perception system distinguishes certain objects from background (from big object group). This gives a possibility to effectively organize UML diagram elements if these elements should be comprehended different from the background.

Perceptual theories give possibility to understand in what way human perceptual system combines objects in subsystems or distinguishes separate elements from background – how a human perceives world around him. Perceptual theory principles can explain why some UML diagrams are better for reading and comprehension than others.

C. Requirements set for layouting of diagram elements

It is possible to define the requirements set for UML diagram elements layouting on perceptual theories basis. This helps to determine UML diagram layouting algorithm tendency. Therefore algorithm gives the opportunity to automate layouting of UML diagram elements and transform the given diagram to its normal form.

Different UML diagram designing software offer different diagram visualization features. For example, some programs have the ability to join links (fig.6 (b)), but in some programs this isn't provided and all links will be separate as it is shown in figure 6 (a). It is understandable that link joining gives more readable UML diagram representation. But not every tool supports this feature, therefore, all requirements connected with tools' functionality will be avoided.

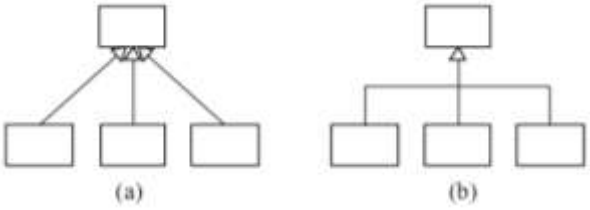

Fig.6. Three separate links (a) and three links joined into one (b) [14]

Previously, all UML diagrams were classified in three subgroups. Diagrams from each group have different structure and construction principles. Different layouting principles can be adjusted for diagrams from different groups. For the first and the second group these principles mostly are the same, but still can differ, because the second group diagrams have several types of nodes or links that must be considered. Every diagram from the third group has its own structure and must be examined separately. Wong and Dabo introduce some requirements that can be used for "pure" graphs and graphs with nodes and links of different types in [14]:

1) Objects that appear larger or different in size from neighbouring can attract attention. Different size can distinguish an object, while uniform sizes can group objects so that they are seen alike.

2) Minimize crossings and bends – the number of edge crossings and bends should be minimized to make edges more continuous and easier to follow. This requirement also is described in Ware et al [21] work.

3) Exploit proximity – generally, diagrams should be compact for easier viewing, but further improvements can be made on the spacing of nodes. Related nodes that should be perceived together should be placed near each other. And, nodes that should not be perceived together should be placed further apart. To enhance grouping, edges should not be too long. These notions are supported by the law of proximity, which is described by Purchase et al in [22].

4) Maximize subset separation – subsets of participants that have little communication should be separated. This is supported by the law of proximity.

5) Avoid overlapping – overlapping should be avoided. Nodes and edges should not overlap other nodes or edges. In figure 7, if overlapping is allowed, (a) would be confusing; because it would not be clear whether box A is connected to box B or directly connected to box C. As a result, it could be perceived as (b), following the law of connectedness, or (c), following the law of continuation.

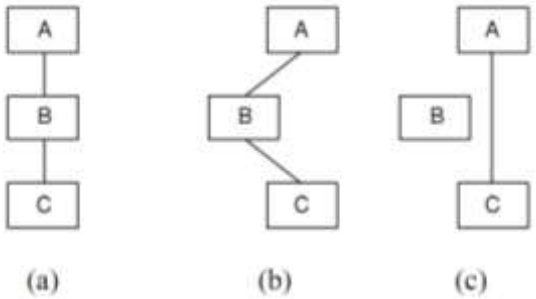

Fig.7. Effect of overlapping

6) Employ symmetry – symmetry within the diagram should be used effectively, since symmetric areas are usually seen as distinct, as well as being "good figure". For example, subclasses of a superclass should be centred. In figure 8, (b) is preferable to (a) because it is symmetric and looks like a stable figure. Research on this requirement can also be found in Purchase [23] research.

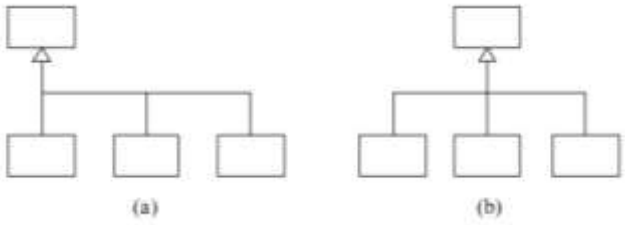

Fig.8. Effect of symmetry

7) Draw links orthogonally – the edges connecting nodes should be orthogonal. This also conforms to the law of orientation.

8) Enhance flow – diagrams can be difficult to read, and it may not be obvious where to start; and, even given some starting point, the next object to look at could be in any direction. Thus, it is useful to control the degrees of freedom, as it is said in Petre studies in [24]. Overall, since people naturally read texts from left to right (in most cultures), and from top to bottom, diagrams should have a similar starting point and subsequent flow. According Ambler research in [25] important classes should be at the top or on the left, and other related classes should be placed below them or towards the right.

Some of the defined requirements conflict with each other (for example, minimize subset separation requirement and exploit proximity requirement). This means that it is essential to define significance for conflicting requirements especially for diagram elements layouting automation; also this ability can be given to user. Authors propose this task for further studies.

The described principles and requirements can be used for creation of an algorithm for diagram elements automated layouting. The authors state three major fields of working with diagram layouting problems. They are:

1) Definition of elements placement during creation of the model.
2) Transformation of one model into another within the one modelling environment;
3) Model export from one tool into another;

Next sections describe each of the stated fields from the perspective of the problems existence and their potential solutions.

4. Layouting Issues within the Working with Models

Diagram element layout is one of the essential factors of UML diagram comprehension. The larger and more complex becomes the diagram due its creation, the more time is spent by its creator on improving diagram element layout. Automation of this process fastens diagram creation and improves readability. Also it can bring unification in diagram layout problem.

Layouting algorithm can be used for model creation before all model elements are included or connected to help its creator to understand the model or after all model elements are included and connected to improve general model readability.

There are two important areas connected with diagram elements layouting: the area of model transportation and its importation.

A. Layouting Issues in the Area of Model Transformation

Many of UML diagram creation tools give opportunity to transform one diagram into another. Usually sequence diagram is converted into communication diagram and backwards. In every tool transformed diagram's logic stays the same but its element layout differs. After transformation, resulting diagram is hardly readable. Layouting algorithm can be automatically used after diagram transformation to improve readability.

Figure 9 shows an example of the sequence diagram transformation into communication one. Resulting communication diagram is hardly readable: messages overlap, links cross nodes and existing nodes are overlapping. All these problems can be solved using layouting algorithm or by manually allocating elements in space. Layouting algorithm usage will partly or fully free the user from the manual elements allocating, this will save diagram creator's time and efforts.

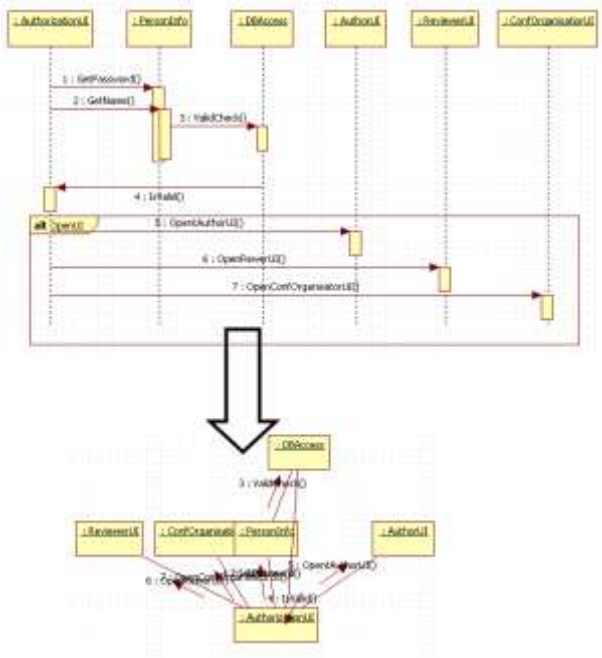

Fig.9. Sequence diagram's transformation in communication diagram in StarUML tool example

Application of the principles of Model Driven Architecture [26] shows more complicated requirements for model transformation where transformations automatically create other types of diagrams, e.g., PSM class diagrams from PIM diagrams, so that automatic layout generation is required more broadly.

B. Layouting Issues in the Area of Model Importation

In some cases there is a need to import UML diagram created in one tool into another. This can be caused by different factors: changing developing team, improving project

created by others, changing developing tools and so on. It is important that diagram should be the same or should have little difference from original after importing.

According to Gulbis in [27] most of the tools provide XMI (XML Metadata Interchange) standard for diagram exporting/importing, but most tools use different XMI standard versions, which are not compatible with each other. If two different tools use different XMI versions, this will cause incorrect UML diagram importation. Gulbis provides experimental results exporting UML class, use case and sequence diagrams from one tool to another using XMI standard provided in [27].

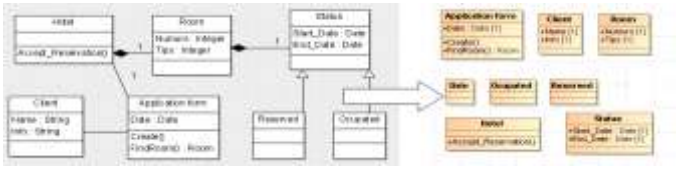

Fig.10. Exporting class diagram from AgroUML to MagicDraw [27]

Figure 10 shows the exporting class diagram from AgroUML tool to MagicDraw tool example – after importing diagram looses all links and diagram elements original coordinates. According to Gulbis in [27] this is common for many tools.

The XMI standard governs only the exchange of abstract syntax (UML domain), coding of diagram elements with their coordinates is an excess of the standard. Most of the UML tools code the layout information in XMI, but do it in a custom way.

Problem with missing links is more important than diagram elements positions. Even if user restores all links from original diagram, he still needs to relocate diagram elements to get original diagram. A manual diagram elements relocating takes effort and consumes time. Automation of this process saves time and efforts. Also if layout of the original diagram was found by layouting algorithm then applying this algorithm to imported diagram with "broken" layout most likely will give the same result.

5. Conclusion

As far as for the task of system implementation, where the formatting guidelines have proved to be a successful method to improve the readability of source code, the increasing success of visual specification languages such as UML for model-driven software development visual guidelines are needed to standardize the presentation and the exchange of modelling diagrams with respect to human communication, understandability and readability. In this article, authors summarized several problems in the area of visual guidelines capturing the aesthetic quality of UML diagrams. We propose these issues as a framework for research on improvement of the aesthetic quality and thus the understandability of UML diagrams. All 13 UML diagram types can be classified according to their structure. This classification helps to distinguish similar diagrams (according to elements' sets and construction principles) and introduce a set of requirements for the effective element layout for particular diagram's groups. This requirement set can be used for algorithm creation and automatic element layouting.

Most requirements introduced by others authors' conflict with each other and must be reconsidered and unified in some manner. Most of requirements can't be thrown away even if they conflict with other requirements, in this case priorities must be used to distinguish, which of them have higher influence on diagram readability improvement.

Automation of diagram element layouting can save creator's time and efforts, and give a better result than manual element distribution. Algorithm for diagram automated layouting brings positive aspects in model creation, transformation and export/import areas. Definition of the algorithm for diagram element layouting in the task of UML diagram readability improvement and development of such algorithm support is stated as a direction of future research of the authors.

Acknowledgments. The research presented in the paper is supported by the research grant No. FLPP-2009/10 of Riga Technical University "Development of Conceptual Model for Transition from Traditional Software Development into MDA-Oriented." The research presented in the paper partly is supported by Grant of Latvian Council of Science No. 09.1245 "Methods, models and tools for developing and governance of agile information systems".

References

[1] Batini C., Furlani L., Nardelly E. *What is a Good Diagram? A Pragmatic Approach.* In Entity-Relationship Approach: The Use of ER Concept in Knowledge Representation, Proceedings of the Fourth International Conference on Entity-Relationship Approach, USA, IEEE Computer Society and North-Holland, pp 312–319, 1985.

[2] Даль В. *Толковый словарь великорусского языка*, 1863. [Online] Available: http://www.rubricon.com.resursi.rtu.lv/tsd_1.asp

[3] Kosak C., Marks J., Shieber S. *Automating the Layout of Network Diagrams with Specified Visual Organization.* IEEE Trans. Systems, Man and Cybernetics 24, 3, pp. 440–454, 1994.

[4] K. Freivalds, P. Kikusts *Optimum Layout Adjustment Supporting Ordering Constraints in Graph-Like Diagram Drawing*. Proc. of the Latvian Academy of Sciences, pp. 43–51, 2001

[5] K. Freivalds, U. Dogrusoz, and P. Kikusts, *Disconnected Graph Layout and the Polyomino Packing Approach*, Proc. of Graph Drawing 2001, Lecture Notes in Computer Science,pp. 378-391, Springer-Verlag, 2002.

[6] Battista G.D., Eades P., Tamassia R., Tollis I.G., *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall. 1999.

[7] Eiglsperger M., Kaufmann M., Siebenhaller M., *A topology-shape-metrics approach for the automatic layout of UML class diagrams.* In: Soft Vis 2003: Proceedings of the 2003 ACM Symposium on Software Visualization, pp.189–198, ACMPress. 2003.

[8] Eichelberger H. *Aesthetics of class diagrams.* VISSOFT 2002: Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis, pp.23–31, 2002.

[9] Dwyer T. *Three dimensional UML using force directed layout.* In: CRPITS 2001: Australian Symposium on Information Visualisation, pp.77–85, Australian Computer Society, Inc, 2001.

[10] Andriyevska A., Dragan. N, Simoes B., Jonathan I. Maletic. *Evaluating UML Class Diagram Layout based on Architectural Importance* Proceedings of the 3rd International Workshop on Visualizing Software for Understanding and Analysis, IEEE Computer Society, 2005.

[11] Eichelberger H., *Automatic layout of UML use case diagrams*, In: Proceedings of the 4th ACM symposium on Software visualization, pp. 105-114, 2008.

[12] Bist G., MacKinnon N., Murphy S. *Sequence diagram presentation in technical documentation.* In: SIGDOC 2004: Proceedings of the 22nd Annual International Conference on Design of Communication, NewYork, NY, USA, pp.128–133, ACMPress. 2004.
[13] Poranen T., Makinen E., Nummenmaa J. *How to draw a sequence diagram.* In: Proceedings of the Eighth Symposium on Programming Languages and Software Tools, SPLST 2003, University of Kuopio, Department of Computer Science, pp.91–102, 2003.
[14] Wong K., Dabo. S *On evaluating the layout of UML diagrams for program comprehension.* Software Qual J, pp. 233–259, 2006.
[15] UML resource page [Online] Available: http://www.uml.org/ [Accessed 26.02.2011].
[16] Tilley S., Huang S., *A qualitative assessment of the efficacy of UML diagrams as a form of graphical Documentation in aiding program understanding,* SIGDOC: Proceedings of the 21st Annual International Conferenceon Documentation, ACMPress, pp.184–191, 2003.
[17] Murdock J. *Normal forms.*2006. [Online] Availiable - http://www.scholarpedia.org/ [Accessed 10.10.2010]
[18] Boff K. R., Kaufman L., Thomas J.P., *Handbook of Perception and Human Performance.* Volume II. 1986.
[19] Goldstein B. *Sensation and perception.* 6thedn. Wadsworth Thomson Learning, 2002.
[20] Kimchi R., Behrmann M., Olson C. *Perceptual organization in vision. Behavioral and Neural Perspectives.* Lawrence Erlbaum, 2003.
[21] Ware C., Purchase H., Colpoys L., McGill M. *Cognitive measurements of graph aesthetics.* Information Visualization, pp.103–110, 2002.
[22] Purchase H. C., McGill M., Colpoys L., Carrington D., *Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study.* CRPITS 2001: Australian Symposium on Information Visualization, pp.129–137, 2001.
[23] Purchase H.C., *Which aesthetic has the greatest effect on human understanding?* In: GD1997: Proceedings of the 5th International Symposiumon Graph Drawing, pp.248–261, Springer-Verlag, 1997.
[24] Petre M., *Why looking isn't always seeing: readership skills and graphical programming.* Communication of the ACM pp.33–44, 1995.
[25] Ambler S.W., *The Elements of UML Style.* Cambridge Univ Press. 2003.
[26] MDA resource page [Online] Available: http://www.omg.org/mda/ [Accessed 26.02.2011].
[27] Gulbis I. *Modeļu datu apmaiņas standartu izpēte modeļu vadāmās programmatūras izstrādes kontekstā*. 2010.

Arthur Galapov received B. Sc. in computer control and computer science in 2009 from Riga Technical University.
Presently, he is a master student of Riga Technical University at the Faculty of Computer Science and Information Technology. He took part in several scientific and industrial projects. He is C/C++ Programmer in the "Accenture", Riga, Latvia.

Oksana Nikiforova has received engineering science doctor's degree (Dr.sc.ing) in information technologies sector (system analysis, modelling and designing, sub-sector) from Riga Technical University, Latvia, in 2001.
She is presently a full professor at the Department of Applied Computer Science of Riga Technical University, where she has worked since 1999. Her current research interests include object-oriented system analysis and modelling especially its issues in the framework of Model Driven Architecture. In these areas she has published extensively and has been awarded several grants. She has participated in and managed several research projects related to the system modelling, analysis and design, as well as participated in several industrial software development projects.
She is a member of RTU Academic Assembly, Council of Faculty of Computer Science and Information Technology, RTU publishing board, RTU Scientific Journal Editorial Board, etc. She is a co-chair of annual workshop of Model Driven Architecture in conjunction with International conference ENASE. She is awarded as RTU Young Scientist of the Year 2009.

Artūrs Galapovs, Oksana Ņikiforova. Diagrammu elementu izvietojums telpā: esošais stāvoklis
Sistēmas modelēšana dod iespēju programmatūras izstrādātajiem saprast lielu sistēmu uzvedību, struktūru, pamatelementus. Viena no mūsdienas lietotām notācijām sistēmas modelēšanai ir vienota modelēšanas valoda (*angl*. Unified Modelling Language – UML). UML diagrammas elementu telpiskā izvietošana spēlē noteicošu lomu programmatūras sistēmas izpratnē. Jo efektīvāka ir elementu izvietošana, jo vienkāršāka ir diagrammas būtības izpratne un jo efektīvāka ir UML diagrammas lietošana. Lai to panāktu diagrammas izstrādātājam ir jāizvieto elementi tā, lai paaugstinātu diagrammas uztveramību. Manuālā elementu izvietošana prasa daudz laika un rezultāts ne vienmēr ir apmierinošs. Šī procesa automatizācija varētu atbrīvot diagrammas izstrādātāju no lieka darba un piedāvāt labāku rezultātu nekā manuāla elementu izvietošana diagrammas konstruēšanas, transformācijas vai eksporta/importa laikā.
Lai izstrādātu algoritmu diagrammas elementu izvietošanas automatizācijai ir jānoteic pēc kādiem principiem ir jāizvieto diagrammas elementus lai paaugstinātu tās uztveramību. Viens no pamatiem, uz kuriem var balstīt diagrammas elementu izvietošanas efektivitātes noteikšanu ir objektu uztveres teorijas. Dotas teorijas izskaidro kādā veidā cilvēka prāts apvieno atsevišķus objektus sistēmās vai otrādi – izdala atsevišķus objektus no kopēja fona.
Dotajā rakstā ir piedāvāta UML diagrammu klasifikācija, kas pamatojas uz diagrammu struktūrām. Dotā klasifikācija dod iespēju izdalīt līdzīgus diagrammu tipus kurām var pielietot vienas un tās pašas prasības elementu izvietošanai telpā. Tas nozīmē, ka pie dotiem diagrammu tipiem var pielietot vienu un to pašu elementu izvietošanas automatizācijas algoritmu.

Артур Галапов, Оксана Никифорова. Размещение элементов UML диаграмм в пространстве: Общая ситуация
UML (англ. Unified Modelling Language) является широко используемым стандартом для моделирования систем, который так же частично может автоматизировать сам процесс разработки. В наши дни инженерам приходится работать с большими и сложными системами, поэтому остро встает вопрос о читабельности диаграммы, так как с разработанными диаграммами работают так же другие разработчики. Важно, чтобы диаграмма была интуитивно понятна тем, кому придется с ней работать. Расположение элементов диаграммы в языке UML в пространстве играет важную роль в вопросе улучшения читабельности диаграммы. Ручное размещение элементов диаграммы в пространстве трудоемкий и долгий процесс, который не всегда дает нужных результатов. Поэтому автоматизация этого процесса может быть решением, которое даст нужный результат за короткое время во время создания, трансформации или экспорта\импорта диаграммы.
Для того, чтобы разработать алгоритм для автоматизации размещения элементов диаграммы в пространстве, нужно определить, по каким принципам нужно размещать элементы диаграммы, чтобы повысить ее читабельность. Для определения требований к взаимному расположению элементов диаграмм были использованы принципы перцептуальной сегрегации и перцептуального объединения, которые объясняют, каким образом сознание человека объединяет отдельные объекты в системы или наоборот, выделяет отдельные объекты из общего фона.
В данной статье предложена классификация UML диаграмм, основываясь на их структуре. Данная классификация дает возможность выделить схожие типы диаграмм, к которым, в свою очередь, можно применить одни и те же требования для взаимного расположения элементов в пространстве. Что так же значит, что к данным типам диаграмм может быть применен один и тот же алгоритм распределения элементов диаграммы в пространстве.