# Several Issues on the Layout of the UML Sequence and Class Diagram

Oksana Nikiforova, Dace Ahilcenoka, Dainis Ungurs, Konstantins Gusarovs, Ludmila Kozacenko
Faculty of Computer Science and Information Technology
Riga Technical University
Riga, Latvia
{oksana.nikiforova, dace.ahilcenoka, dainis.ungurs, konstantins.gusarovs, ludmila.kozacenko}@rtu.lv

***Abstract* — Models are widely used and are one of the advanced tools of software engineering. Therefore, it is very important that the models and diagrams are well built not only considering their content, but also how they visually represent information, how they are layout. Layout is an important factor considering readability and comprehensibility of a diagram. Providing manual diagram layout is time consuming; it can also be ineffective; therefore, this paper is a research about diagram automatic layout. UML provides a variety of diagrams, which covers all of the system development life cycle steps. The most important UML diagrams are class and sequence diagrams, because they are the main diagrams to present system structure and behavior. We analyze existing layout techniques and algorithms, offer new ones and evaluate them regarding their applicability to class and sequence diagram layout in different modeling tools, how they comply with layout criteria.**

***Keywords – UML class diagram; UML sequence diagram; layout algorithm; BrainTool.***

## I. INTRODUCTION

One of the tasks of software development is to present different aspects of the system before developing the software solution for that system. To solve this task, system modeling became one of the important activities during software development. Models are useful for understanding problems, communicating with everyone involved within the project (customers, domain experts, analysts, designers, etc.), modeling enterprises, preparing documentation and designing programs and databases. Modeling promotes better understanding of requirements, more clear designs and more maintainable systems. Graphical models help to provide a common base for system developers at different levels of system domain and are used at different stages of system abstraction. It is specially pointed to such standardized modeling mean as Unified Modeling Language (UML) [1].

The graphical aspect of modeling language turns developers to an intuitive language semantics and perceptible location of model elements on the diagram. Thus, modelers have to decide two main tasks during creation of the diagram: to think of how to present system functionality by diagram elements and to invent an optimal placement of diagram boxes and wires. Thus, the systematic approach to elements placement within the diagram, which is specified as a task of diagram layout, plays an important role in completing the task of system modeling. This paper tries to solve the problem of diagram layout in correspondence with the most used UML diagrams, namely the UML class diagram and the UML sequence diagram. The goal of the research is to offer layout algorithms for both diagrams, to implement the presenting algorithms within the BrainTool [2] modelling tool, which gives an ability to generate UML diagrams from the two-hemisphere model [3].

The paper is structured as follows. The next sections introduce requirement set to layout the UML sequence and class diagrams. The algorithm based on the defined requirement sets is described in the second and the third section. The fourth section gives a brief overview of the related work and compares our solution with the existing ones. We discuss about the present research and state the direction for the future in the conclusion of the paper.

## II. LAYOUT ALGORITHM FOR THE UML SEQUENCE DIAGRAM

Basically, the software system development starts with the business information gathering and presenting it in the form suitable for further software system modeling. Then, this presentation of business information has to be transformed into the model, which in object-oriented manner for software development requires to present objects to interact in the form of UML sequence diagram [1]. It shows objects, their lifelines and messages to be sent by objects-senders and performed by object-receivers and is used to present dynamic aspect of the system. The dynamic of interactions is defined by an ordering of the messages. It serves as a basis for definition of operations performed by objects to be grouped into classes, as well as to present and to verify a dynamic aspect of class state transition. UML sequence diagram is a popular notation to specify scenarios of the processing of operations as its clear graphical layout gives an immediate intuitive understanding of the system behavior. UML sequence diagram is stated as one of the ambiguous UML diagrams, with an implicit and informal semantics that designers can give to basic sequence diagram as a result of this conflict.

The time aspect plays the most important role and helps to organize messages in correct sequences. Vertical axis is used to display time, the beginning of the diagram is at the top and it is read downwards. Sequence diagram can consist of many different elements; however, the authors will use only those, which can be acquired from the two-hemisphere model, which is a kind of initial presentation of the problem domain in the model form, which consist of process model interrelated with conceptual model. It is possible to generate UML sequence and class diagram from the two-hemisphere model based on the direct transformation of diagrams, which are already explained in [3][4]. To enable implementation of the model transformation by a tool, it is necessary to have an algorithm for element placement after the transformation execution.

## *A. Layout Requirements for the UML Sequence Diagram*

Table 1 shows the list of criteria for layout the elements of the UML sequence diagram in descending order of their importance. Criteria are marked with SD identifier. General layout criteria result from the theory of perception [5]. Specific diagram like the UML sequence diagram has additional criteria, e.g., "slidability". There are six perceptual principles referring to organization of diagram elements, when the elements are considered as a group [6]. These principles are acquired from Gestalt theory [5]. There are three more principles related to perceptual element segregation. All of these Gestalt theory principles are considered as aesthetic criteria. General aesthetic criteria are widely discussed in [7][8][9].

TABLE I. CRITERIA FOR LAYOUT OF THE UML SEQUENCE DIAGRAM

| ID | Name of criterion | Description |
|---|---|---|
| SD0 | Precise sequence of messages | Notational convention of the UML requires to display messages in the order they are being sent. |
| SD1 | Avoid object and lifeline overlapping | When objects or lifelines are overlapping it is hard or sometimes impossible to read the diagram. |
| SD2 | Elements to be arranged orthogonally | Sequence diagram is an example of orthogonal diagram - message arrows are situated horizontally (typically) and lifelines - vertically. |
| SD3 | Diagram flow | It is very important to layout elements by creating obvious flow - visible start and end of the diagram, easier to follow the elements and read the diagram. The first message is located at the top left corner of sequence diagram. |
| SD4 | Minimize crossings | In the sequence diagram message arrows should not cross at all, therefore with crossings is understood message arrow crossings over lifelines and number of this kind of crossings should be reduced. |
| SD5 | Message arrow length minimization | To make the diagram more comprehensible and the area smaller, the message arrow length should be minimized |
| SD6 | Reduction of long message arrow number | It is difficult to follow long message arrows, so they should be as few as possible. |
| SD7 | Minimize longest message arrow length | The longest message arrow should be shortened if possible, e.g., placing elements closer. |
| SD8 | Uniform message arrow length | Message arrows with similar length make diagram more understandable. Similar arrow length is also needed to fulfill the "slidability" criteria. |
| SD9 | Improve "slidability" | "Slidability" is an aesthetic criteria for better clearness, particularly important in bigger sequence diagrams, where the whole diagram fails to fit in one screen. |

A sequence diagram is specific in its visual presentation. All the objects are allocated horizontally at the top of the diagram and the life lines are drawn vertically top-down. Therefore, the criteria for the UML sequence diagram should be carefully selected or even modified, so that they could be applied. For example, one specific criterion for sequence diagram is correct sequence of messages, which is the meaning of this diagram. Poranen et al. [10] and Wong and Sun [6] have identified the criteria specific for sequence diagrams.

## *B. Basic Principles of the Layout Algorithm*

Considering the specificity of sequence diagram the authors propose to use an algorithm, which is based on topology-shape-metrics planarization step and uses one principle of force-directed approach – object tends to attract those objects, with which it communicates. The algorithm places the elements possibly close and tries to arrange communicating participants beside based on priorities. Priorities are calculated considering object attraction forces – as more messages between elements as higher priority for them to be beside. The layout algorithm calculates the distance between the elements considering lengths of messages and class object names. Algorithm places elements as close as possible by taking into account the diagram flow (e.g., interacting objects are being placed beside if possible). The pseudo-code of the layout algorithm implemented is the following [11]:

```
func layout_SD(Sequence_object obj[], Message msg[],
 Sequence_object result[])
  //determines the first object
  for i=0 to obj.size do
    if msg[i].sequenceNum = 1
        add to result object sequence msg[i].sender
  //determines priorities of other objects
  for i=0 to obj.size do
     for j=0 to msg.size do
        if (msg[i].sender = result[result.size-1] AND
        msg[i].receiver!= result[result.size-1])
            for k=0 to obj.size do
                 if msg[j].receiver = obj[k]
                   obj[k].priority+1
        if (msg[i].receiver = result[result.size-1] AND
        msg[i].sender != result[result.size-1])
            for k=0 to obj.size do
                 if msg[j].sender = obj[k]
                   obj[k].priority+1
     //excludes from not ordered obj[] element, that
     //was last added to result[]
     for j=0 to obj.size do
        if (obj[j]= result[result.size-1])
           delete objekts[j]
     //adds highest priority object to result[]
     for j=0 to obj.size do
        if (obj[j].priority=highest priority
          adds to result[] obj[j]
  //adds coordinates to objects
  lastXcoordinate=X_START_COORDINATE
  for i=0 to result.size do
     result[j].y=Y_ START_COORDINATE
     //MINIMAL_SPACING needs to be computed considering
     //last added object size, current object size and
     //maximal between those two object sent and received
     //message label length
     result[j].x= lastXcoordinate + MINIMAL_SPACING
     lastXcoordinate = result[j].x
```

Figure 1 shows an example of "bad" layout of the UML sequence diagram and the corresponding good layout of the same object interaction. The algorithm is implemented in BrainTool, which serves for generating UML diagrams from the two-hemisphere model mentioned above.

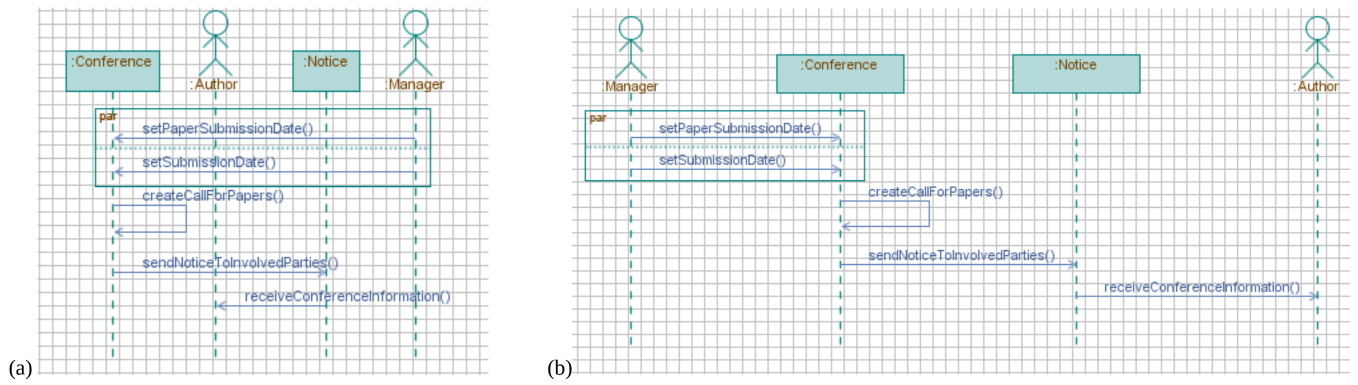

Figure 1. Example of “bad” (a) and “good” (b) layout of the UML sequence diagram.

## III. Layout Algorithm for the UML Class Diagram

The UML class diagram describes system’s structure by showing its classes with the methods and attributes, and relations between classes. The visual presentation of the UML class diagram lookes like graph with vertexes and edges, but due to class diagrams ability to present different types of relationships between classes, the diagram is classified as a graph with specific constructions of arcs of different types.

### *A. Layout Requirements for the UML Class Diagram*

There is no set standard for the location of classes. It is generally agreed upon to place the most important objects at top left and read the diagram to the right and downwards [6], however, by not following this rule would not make the diagram less readable.

By analogy with the UML sequence diagram, layout criteria for the UML class diagram result from the theory of perception. The same as for sequence diagram, the layout algorithm for class diagram should take into consideration all the Gestalt theory principles described above.

In addition, it is possible to define requirements’ set for UML class diagram elements’ layout on basis of perceptual theory. This helps to determine UML diagram’s layout algorithm’s tendency. Therefore, an algorithm provides the opportunity to automate layout of UML diagram’s elements and transform given diagram to its normal form.

Some of defined requirements conflict with each other (for example, minimizing the subset separation requirements and exploiting the proximity requirement). This means that it is essential to define significance for conflicting requirements especially for diagrams’ elements layout automation; also this ability can be given to user. We leave this task and discovering of new requirements for diagrams’ elements layout for further studies.

However, it is up to the creator of the layout algorithm to decide which criterion is more important. All the described principles and requirements can be used for creation of algorithm for diagram’s elements’ automated layout. Table 2 shows the list of criteria for the UML class diagram layout. Criteria are marked with CD identifier.

TABLE II. Criteria for Layout of the UML Class Diagram

| ID | Name of criterion | Description |
|---|---|---|
| CD0 | Join inheritance arcs | Joining inheritance arcs provide a more understandable structure and suggest hierarchy. It also decreases the amount of connections to a class, which can make it easier to view. |
| CD1 | Ensure association representation | There are several ways to represent associations, depending on how much information is shown. |
| CD2 | Employ selectivity | Some information contained in a class or relationship can be less useful than other, so displaying only the useful information can help the understandability of the diagram. |
| CD3 | Use colors | Many people are sensitive to colors [5]. This can be used to visually group classes. |
| CD4 | Minimize crossings and bends | Crossings and bends can make it harder to distinguish what classes a relationship connects. |
| CD5 | Center parents or children | Centering parents or children can visually group them together. |
| CD6 | Reduce length of relationships | Shorter relationships help decrease the size of the diagram and make it easier to view. |
| CD7 | Ensure inheritance direction | It is generally agreed upon, that child classes should be placed below parent class [8]. This helps display the hierarchy. |
| CD8 | Avoid overlapping | Overlapping can cause loss of data and remove the representation of object shapes. All this leads to less readable diagrams. |
| CD9 | Employ symmetry | Symmetry can improve the readability of a diagram. |
| CD10 | Employ orientation | It is advisable to layout diagrams in a way, to read them from top to bottom and from left to right. This is more common in most countries and helps to guide the flow of information. |
| CD11 | Employ orthogonality | Orthogonal relationships are easier to follow than bent or straight lines and help avoid overlapping. |
| CD12 | Place labels horizontally | Placing all the labels horizontally helps readability of the diagram. |
| CD13 | Place associations horizontally | Associations should be placed horizontally if possible. It helps readability of the diagram as well as placement of labels. |

## B. Basic Principles of the Layout Algorithm

The layout algorithm operates in four major steps [12]. Prior to these steps, algorithm gathers data on all the classes and their relationships in the diagram and places them in specific data types and constructs, for easier usage.

The pseudo-code of the layout algorithm implemented is the following [12]:

```
func layout_CD(Class_object obj[], Relationship rel[],
int iteration_count)
for n=0 to iteration_count do
   //Assign each class a score
   for i=0 to obj.size do
      set obj[i].score based on rel[obj[i]]
   //Group the classes
   while !obj[].isEmpty
      for i=0 to obj.size do
         find obj[i].score = highest
         set max=i
         set max_group_level++
      set obj[max].grouplevel
      remove obj[max] from obj[]
      for i=0 to obj.size do
         find obj[i] with rel[obj[max]]
         set obj[i].grouplevel = obj[max].grouplevel
         remove obj[i] from obj[]
   for i=0 to max_group_level
      add obj.grouplevel[i] to group[i]
   //Layout the groups
   for i=0 to group.size
      for j to group[i].obj.size
         if rel[obj[j]] is optimal do
            set obj[j].coordX AND obj[j].coordY
   //once iteration is completed, repeat,
   //assuming created groups as classes
```

During the first step, each class is assigned a score. This score is calculated based on how many relationships a class has, as well as the type of these relationships. Additionally, class content is taken into account, such as attribute and method count.

In the second step, all the classes are divided into small groups. The groups are created around the classes with the highest scores. Each group contains classes' no more than two relationships away from the main class of the group. This ensures each group is compact and contains classes with similar content. As a special condition, parent class that has generalization relationships will always be the main class of a group. All the child classes will be part of it.

The third step covers the layout of individual groups. Since the diagram is now divided into many small clusters of classes, each group contains a small and limited amount of classes. This limit can be set by the user to personalize the workflow of algorithm. Because the amount of cases is small, a simple layout can be applied, by checking the type of relationship classes have and placing them accordingly, to suit various layout criteria.

Step four involves returning and re-doing steps one to three, treating the newly created and laid out class groups as standalone classes. Because the classes are generally drawn as rectangles, so a group of classes can also be combined and displayed in a similar way. An already implemented approach that remotely resembles this is structured class notation [1].

In order to successfully implement the steps described above, a specific approach is used. All the classes and in later iterations- class groups, are placed in a container object. This object contains all the required data for the layout- class coordinates, width, height and score. Because it can contains a single class and a group of classes, the algorithm only needs to iterate trough the same object type. This improves the workflow of the algorithm.

Figure 2 shows an example of "bad" layout of the UML class diagram and the corresponding good layout of the same class structure. The class diagram consists only from seven classes, but still it is possible to demonstrate the ability of the algorithm to layout the diagram.

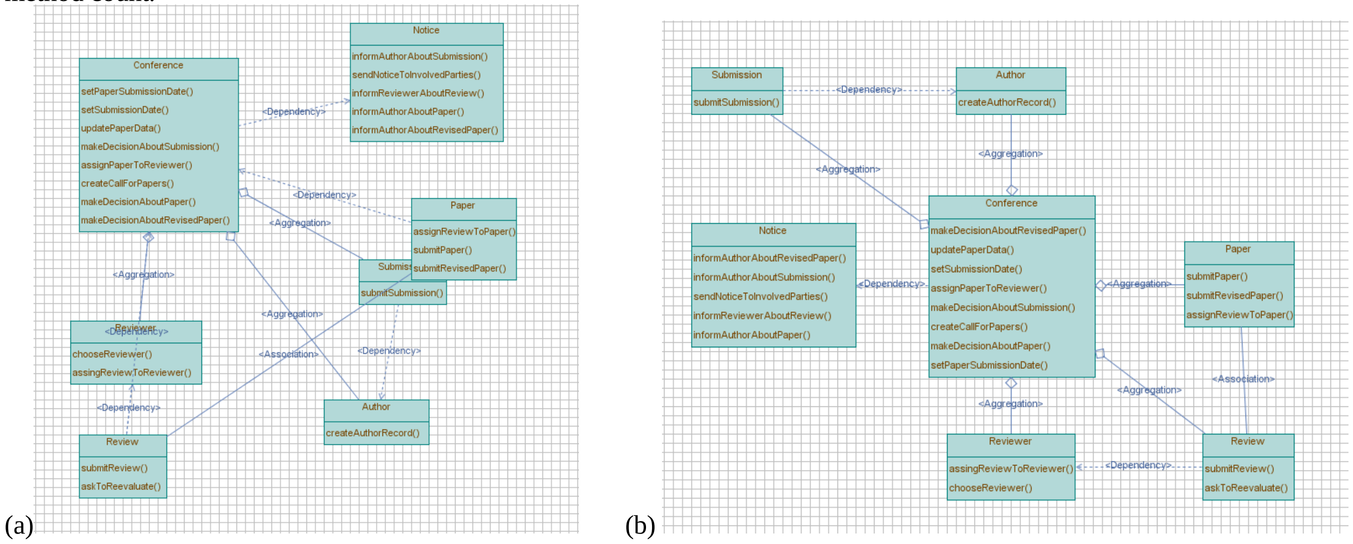

Figure 2. Example of "bad" (a) and "good" (b) layout of the UML class diagram

## IV. Related Work and Evaluation of the Result

The problem of automatic UML diagram layout still exists and it is widely discussed in relation to class and sequence diagrams. The cause of the layout problem is that algorithms are not well suited for each diagram type and there are many different aesthetic criteria to comply with. Some of the criteria are easier to implement than others, for example SD1, SD2, SD4-SD8 [10]. Another problem in automatic layout is that many of the aesthetic criteria are conflicting, e.g., message arrow length minimization (SD5) and minimization of crossings (SD4), and because reducing message arrow length is more likely to cause more crossings. The authors of [10] mention that the optimal layout is algorithmically complicated challenge, which is one more problem to automatic layout, for example an optimal linear layout problem is considered as NP-complete problem [13].

Since there are many different layout algorithms a solution can be found by studying different possibilities to tailor algorithm to specific problem or combine several of them to get the expected results. Diagram layout algorithms are based on graph theory and graph layout algorithms [14]. Algorithms can be divided into approaches, where the most used ones are topology-shape-metrics, hierarchical, visibility, divide and conquer, force-directed approaches; they are described in [7]. Genetic algorithms can also be used in diagram layout.

Topology-shape-metrics approach is one of the most used one [15]. The approach is suitable for orthogonal graphs and it supports many different aesthetic criteria [7]. Eichelberger and Schmid [9] mention that the algorithms of this approach have been used to layout UML class diagrams and are implemented in such tools as GoVisual [16] and yWorksUML [17]. The approach has three main steps, namely, planarization, orthogonalization and compaction, which are well described by di Batista et al. [7].

Hierarchical approach, also called Sugiyama approach, is also used in UML class diagram automatic layout [18]. This approach is suitable for directed acyclic graphs - more or less hierarchic graphs, which is not the sequence diagram case.

Visibility approach is the general approach suitable for various types of graphs. It has been used in entity-relationship diagram layout by Tamassia [19]. The approach also has three main steps, as mentioned in [9] and [7]. This approach can be put in the middle between both previously described approaches. Having studied this approach more closely we can conclude that this approach is less suitable for the sequence diagram than topology-shape-metrics because of its second and third steps.

Divide and conquer approach first divides graph in parts, arranges elements and then merges these parts together [7]. Regarding to sequence diagram, this approach is only suitable to diagrams with separable subsets therefore not suitable for all kinds of sequence diagrams.

Force-directed approach is suitable for undirected graphs [7]. The force-directed approach simulates a physical system of forces, where a system tries to achieve the state of minimum energy. One of main criteria in this approach is minimization of crossings, which is not the most important criteria for sequence diagrams.

There is a wide range of genetic algorithms and they can be used for various purposes, as it was mentioned by Galapovs and Nikiforova in [20]. Genetic algorithms simulate processes from nature, like mutations crossover and selection. Genetic algorithms were used in [21] for class diagram layout and according to the research results these algorithms are time consuming (20 minutes for 17 class layout).

Authors compared the relevance of each algorithm for the sequence diagram to genetic algorithms, topology-shape-metrics and force-directed approach algorithms proved to be theoretically most suitable according to how they meet the sequence diagram criteria. Other approaches are not considered to be suitable at all because they either do not consider the right order of the priorities of criteria or they are not suitable for such diagram/graph type (e.g., they are tailored for undirected, acyclic types of graphs, but the sequence diagram is directed and cyclic).

There are several tools that provide automatic diagram layout, e.g., Borland Together [22] supports automatic UML sequence diagram layout, but uses lawless set of layout criteria while Rational Rose [23] supports UML class, but does not support sequence diagram layout. Sparx Enterprise Architect [24] and Visual Paradigm [25] are tools that also provides automatic UML sequence diagram layout, however, it does not satisfy all the mentioned criteria of layout [11].

Table 3 shows how different criteria of the UML sequence diagram layout are supported by different algorithms and how they are implemented in UML modeling tools. The evaluation “Yes/No” means that criterion is/is not supported. The evaluation “partly” means that criterion is not supported completely, only part of the criterion is implemented. The evaluation “adjustable” means that criterion can be implemented by the algorithm.

The same evaluation for criteria supporting in different modeling tools according the layout of class diagram is shown in Table 4.

Researches also have been made on other types of UML diagrams. Eichelberger and Schmid [9] give researches on automatic layout of UML use case diagrams. Bist et al. presented an approach to draw sequence diagrams in technical documentation to ease communication between project members [26]. Poranen et al. proposed various criteria for drawing a sequence diagram based on traditional graph drawing aesthetics and the special nature of sequence diagrams [10]. Wong and Dabo give requirement set based on cognitive science for sequence and class diagrams, which can help to improve diagrams’ readability [27].

The KIELER project [28] evaluated the usage of automatic layout and structure-based editing in the context of statecharts. It provided a platform for exploring layout alternatives and has been used for cognitive experiments evaluating established and novel modeling paradigms. However, it was rather limited in its scope and applicability.

TABLE III. CRITERIA EVALUATION FOR LAYOUT OF THE UML SEQUENCE DIAGRAM

Abbreviations used in the table are the following: TSMA – Topology-Shape-Metrics Approach, HA – Hierarchical Approach, VA – Visualization Approach, DCA – Divide and conquer approach , FDA – Force-Directed Approach, MSA – Multi-Scale Algorithms, GA – Genetic Algorithms, EA – Enterprise Architect, T – Together, VP – Visual Paradigm, BT – BrainTool, adj – adjustable.

| ID | Name of criterion | TSMA | HA | VA | DCA | FDA | MSA | GA | EA | T | VP | BT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SD0 | Precise sequence of messages | yes | yes | yes | yes | yes | yes | adj | partly | partly | partly | yes |
| SD1 | Avoid object and lifeline overlapping | yes | no | no | no | no | adj | adj | yes | yes | yes | yes |
| SD2 | Elements need to be arranged orthogonally | no | no | no | no | no | no | adj | partly | no | yes | yes |
| SD3 | Diagram flow | yes | yes | yes | adj | yes | no | adj | no | no | no | yes |
| SD4 | Minimize crossings | adj | adj | adj | adj | adj | adj | adj | no | partly | no | yes |
| SD5 | Message arrow length minimization | adj | adj | adj | adj | adj | adj | adj | no | no | no | yes |
| SD6 | Reduction of long message arrow number | adj | adj | adj | adj | adj | adj | adj | no | no | no | yes |
| SD7 | Minimize longest message arrow length | yes | yes | no | adj | yes | no | adj | no | no | partly | yes |
| SD8 | Uniform message arrow length | no | no | no | no | no | no | adj | no | no | no | no |
| SD9 | Improve “slidability” | no | no | no | yes | yes | no | adj | no | no | no | no |

TABLE IV. CRITERIA EVALUATION FOR LAYOUT OF THE UML CLASS DIAGRAM

| Algorithms | CD1 | CD2 | CD3 | CD4 | CD5 | CD6 | CD7 | CD8 | CD9 | CD10 | CD11 | CD12 | CD 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sparx Enterprise Architect 11** | | | | | | | | | | | | | |
| Ring | No | Yes | Yes | Yes | Poor | No | Medium | No | Poor | No | Yes | No | Yes |
| Ellipse | No | Yes | Yes | Yes | Poor | No | Medium | No | Poor | No | Yes | No | Yes |
| Box | No | Yes | Yes | Yes | Poor | No | Poor | No | Medium | No | Yes | No | Yes |
| Page | No | Yes | Yes | Yes | Very Poor | No | Medium | No | Poor | No | Yes | No | Yes |
| Di-graph | No | Yes | Yes | Yes | Good | No | Medium | Yes | Medium | Poor | Yes | No | Yes |
| Spring | No | Yes | Yes | Yes | Medium | No | Good | No | Poor | No | Yes | No | Yes |
| Right to left | No | Yes | Yes | Yes | Very Good | No | Medium | Yes | Good | Poor | Yes | Yes | Yes |
| **Visual Paradigm 11** | | | | | | | | | | | | | |
| Automatic | No | Yes | Yes | Man. | Very Good | No | Medium | No | Very Good | Medium | Yes | Yes | Yes |
| Hierarchical | Yes | Yes | Yes | Man. | Medium | No | Very Poor | Yes | Good | Medium | Yes | Medium | Yes |
| Orthogonal | No | Yes | Yes | Man. | Good | No | Medium | No | Very Good | Medium | Yes | Yes | Yes |
| Ring | No | Yes | Yes | Man. | Medium | No | Good | No | Good | No | Yes | No | Yes |
| Organic | No | Yes | Yes | Man. | Medium | No | Good | No | Good | No | Yes | No | Yes |
| Compact | No | Yes | Yes | Man. | Medium | No | Good | No | Good | No | Yes | No | Yes |
| **MagicDraw 18.0 beta** | | | | | | | | | | | | | |
| Class diagram | Yes | Yes | Yes | Yes | Good | No | Medium | Yes | Good | No | Yes | Yes | Yes |
| Hierarchycal | Yes | Yes | Yes | Yes | Good | No | Medium | Yes | Medium | No | Yes | Yes | Yes |
| Orthogonal | No | Yes | Yes | Yes | Good | No | Poor | No | Good | No | Yes | Yes | Yes |
| Organic | No | Yes | Yes | Yes | Medium | No | Good | No | Good | No | Yes | Yes | Yes |
| Circular | No | Yes | Yes | Yes | Medium | No | Poor | No | Good | No | Yes | Yes | Yes |
| **Braintool** | | | | | | | | | | | | | |
| Organic | No | Yes | Yes | Yes | Good | No | Good | No | Poor | Poor | Yes | No | Yes |
| Compact | No | Yes | Yes | Yes | Medium | No | Good | No | Medium | Medium | Yes | Good | Yes |
| Modular | No | Yes | Yes | Yes | Good | Yes | Good | Yes | Medium | Good | Yes | No | Yes |

Purchase et al. analyzed graph layout aesthetics in UML diagrams, focusing on user preferences, and conducted empirical studies of human comprehension to validate those aesthetic criteria and rank their effect [8]. They also compared various UML notations, and suggested which notations are more understandable [29].

Since there are so many criteria, with some conflicting with each other, software engineers and tool designers are often overwhelmed and confused on choosing the appropriate algorithm to use. The result of the experiments with diagram import/ export and evaluation of their layout in several modeling tools shows that there are still problems with optimal allocation of diagrams elements. And still the problem is not solved. Therefore, we can assume that the algorithms offered and implemented in BrainTool is a step forward in the evolution of the UML diagram layout.

## V. Conclusion

As mentioned in the introduction of the paper, the task of element placement during system modeling has an impact on better understanding of system model and more effective usage of them during development of the system. Nowadays, one of the leaders in system development is object oriented manner of software development and object oriented system modeling has its own way for presentation of different aspects of the system. Therefore, we focused on the problem of diagram layout creating UML diagrams, which is declared as a standard for presentation of software system model and provides a notation, which grows from analysis through design into implementation in object oriented programming languages.

As a notation of system modeling for different aspects of the system, UML introduces 14 types of diagrams, which can describe system from different points of view. However, Ambler stress yet in 2004 that the UML class, sequence and activity diagrams are considered more important than others [30]. And since that time, after 10 years, still the state of the art is not changed in the importance and popularity of the UML diagrams. Nowadays, commonly used UML diagrams in software development projects still are the UML class and sequence diagrams [31][32]. So far, we presented the layout algorithm and its application for UML class and sequence diagram and demonstrates the application of the algorithms in the model transformation tool – BrainTool.

The layout algorithms we offered for the two UML diagrams, namely, sequence and class, satisfy the most criteria stated for diagram layout, which are defined by different authors. In the case with the UML sequence diagram the algorithm support 8 criteria from 10 stated, whereas the best result is 4 criteria for other algorithms and two criteria for the modelling tool. In the case with the UML class diagram, the evaluation of the algorithm offered and implemented by BrainTool is also the same obvious.

## Acknowledgment

The research presented in the paper is supported by Latvian Council of Science, No. 342/2012 "Development of Models and Methods Based on Distributed Artificial Intelligence, Knowledge Management and Advanced Web Technologies".

## References

[1] Unified Modeling Language: superstructure v.2.2, OMG. Available: http://www.omg.org/spec/UML/2.2/Superstructure [retrieved: August, 2014].

[2] BrainTool. Available at http://braintool.rtu.lv/ [retrieved: August, 2014]

[3] O. Nikiforova and M. Kirikova, “Two-hemisphere model drivenapproach: engineering based software development,” The 16th International Conference Advanced Information Systems Engineering, A. Persson and J. Stirna, Eds. BerlinHeidelberg: Springer-Verlag, LNCS 3084, 09.2004, pp. 219-233.

[4] Nikiforova O., Kozacenko, and D. Ahilcenoka. “Two-Hemisphere Model Based Approach to Modelling of Object Interaction,” Proceedings of the Eight International Conference on Software Engineering Advances, Mannaert H. et al. (Eds), IARIA ©, Venice, Italy, October 28-November 1, 2013, pp. 605-611.

[5] B.E. Goldstein, Sensation and Perception. Wadsworth, 2002.

[6] K. Wong and D. Sun “On evaluating the layout of UML diagrams for program comprehension,” IWPC 2005, 13th International Workshop on Program Comprehension, May 15-16, 2005, St. Louis, Missouri, USA. IEEE Computer Society 2005, pp. 317-326.

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

[8] H. C. Purchase, J-A. Allder, and D. Carrington, "Graph Layout Aesthetics in UML Diagrams: User Preferences," Journal of Graph Algorithms and Applications, vol. 6, no. 3, 2002, pp. 255-279. [Online]. Available: Universitat Trier, http://www.informatik.uni-trier.de [retrieved: August, 2014].

[9] H. Eichelberger and K. Schmid, "Guidelines on the aesthetic quality of UML class diagrams." In Information and Software Technology, vol. 51, no. 12, 2009, pp.1686-1698. [Available: ScienceDirect, http://www.sciencedirect.com. [retrieved: August, 2014].

[10] T. Poranen, E. Makinen, and J. Nummenmaa “How to Draw a Sequence Diagram,” SPLST'03 Proceedings of the Eighth Symposium on Programming Languages and Software Tools, June 17-18, 2003, Kuopio, Finland. University of Kuopio, Department of Computer Science 2003, pp. 91-102.

[11] D. Ahilcenoka, Development of the Layout Algorithm of the UML Sequence Diagram, Master Thesis, Riga Technical University, 2014.

[12] D. Ungurs, Development of the Layout Algorithm of the UML Class Diagram, Master Thesis, Riga Technical University, 2014.

[13] M. Garey and D. Johnson, Computers and intractability - A Guide To The Theory Of NP- Completeness. W.H. FREEMAN AND COMPANY, 1991.

[14] K. Freivalds, U. Dogrusoz, and P. Kikusts, “Disconnected Graph Layout and the Polyomino Packing Approach,” GD 2001 9th International Symposium on Graph Drawing, September 23-26, 2001, Vienna, Austria. Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 378-391.

[15] J. Sun, Automatic, Orthogonal Graph Layout, Project work, Hamburg University of Technology, 2007.

[16] Oreas optimization, research and software, GoVisual Diagram Editor. Available: http://www.oreas.com/gde_en.php. [retrieved: August, 2014]

[17] yWorks, Automatic Layout of Networks and Diagrams. Available: http://www.yworks.com/en/products_yfiles_practicalinfo_gallery.html [retrieved: August, 2014].

[18] J. Seemann, “Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object-Oriented Software Diagrams,” GD '97, Graph Drawing, 5th International Symposium, September 18-20, 1997, Rome, Italy. New York: Springer Verlag, 1997, pp. 415-424.

[19] R. Tamassia, “New Layout Techniques for Entity-Relationship Diagrams,” Proceedings of the Fourth International Conference on Entity-Relationship Approach October 29-30, 1985, Chicago, Illinois, USA. IEEE Computer Society and North-Holland, 1985, pp. 304-311.

[20] M. Mitchell, An Introduction to Genetic Algorithms. A Bradford Book, 1999.

[21] A. Galapovs and O. Nikiforova, “Several Issues on the Definition of Algorithm for the Layout of the UML Class Diagram,” 3rd International Workshop on Model Driven Architecture and Modeling Driven Software Development (MDA & MDSD 2011) in conjinction with the 6th International Conference on Evaluation of Novel Approaches

to Software Engineering, June 8-11, 2011, Beijing, China. SciTePress Digital Library 2011, pp. 68-78.

[22] Borland a micro focus company, Borland Together. Available: http://www.borland.com/products/Together/. [retrieved: August, 2014].

[23] IBM, Rational Rose product family. Available: http://www-01.ibm.com/software/awdtools/developer/rose/. [retrieved: August, 2014]

[24] Visual Paradigm "Drawing activity diagrams". Available: http://www.visualparadigm.com/support/documents/vpumluserguide/94/200/6713_drawingactiv.html [retrieved: August, 2014].

[25] Sparx systems, "Enterprise Architect". Available: http://www.sparxsystems.com.au/ [retrieved: August, 2014].

[26] G. Bist, N. MacKinnon, and S. Murphy, "Sequence diagram presentation in technical documentation," SIGDOC 2004: Proceedings of the 22nd Annual International Conference on Design of Communication, NewYork, NY, USA, ACMPress, 2004, pp. 128–133.

[27] K. Wong and S. Dabo, "On evaluating the layout of UML diagrams for program comprehension," Software Quality Journal, 2006, pp. 233–259.

[28] KIELER project, the Kiel Integrated Environment for Layout Eclipse Rich Client [Online] Available http://www.informatik.uni-kiel.de/rtsys/kieler [retrieved: August, 2014]

[29] H.C. Purchase, M. McGill, L. Colpoys, and D. Carrington "Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study," CRPITS 2001: Australian Symposium on Information Visualisation, Australian Computer Society, Inc., 2001, pp. 129–137.

[30] S. W. Ambler. The Object Primer: Agile Model-Driven Development with UML 2.0. Third Edition. Cambridge, UK: Cambridge University Press, 2004. 572 p. ISBN 978-0521540186.

[31] Runtime Verification: First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings (Lecture Notes in Computer Science / Programming and Software Engineering), Barringer et al. (eds.), 2010.

[32] L. T. Yang, E. Syukur, and S. W. Loke Handbook on Mobile and Ubiquitous Computing: Status and Perspective, CRC Press, 2012.