Software Design Document: Team 2.1

Contents

1 Change History

Version	Authors	Description	Date
1	Team 2.1 [1]	Initial version	13.05

Table 1: Change History

2 Introduction

2.1 Purpose of the System

Supplying two VANTED plugins, enabling users to enhance layout performance and quality using the supplied multilevel framework and using the stress minimization layout algorithm for calculating graph layouts.

2.2 Design Goals

This document gives a detailed overview about the architecture of the software and furthermore contains all information that is necessary for the implementation process.
The software shall supply the following functionality:

- The multilevel framework
- The stress minimization layout algorithm

2.3 Definitions, Acronyms, and Abbreviations

- VANTED: Visualisation and Analysis of Networks conTaining Experimental Data
- SRS: Software Requirements Specification ("Pflichtenheft")
- SDD: Software Design Document
- GUI: Graphical User Interface

2.4 Reference

INT Introductory lecture on 16/04.

SUB Document "Softwareprojekt2019_Gruppe2Thema.pdf"

ME1 Supervisor meeting 1

ME2 Supervisor meeting 2

ME3 Supervisor meeting 3

EM1 e-Mail from Karsten Klein from 29/04/2019 with subject: "Re: Weitere Fragen Software Projekt"

EM2 e-Mail from Michael Aichem from 16/05/2019 with subject: "Antworten auf eure Fragen vom Betreuermeeting"

IM1 Internal meeting 1

PA1 "COAST: A Convex Optimization Approach to Stress-Based Embedding" by Emden R. Gansner, Yifan Hu, and Shankar Krishnan at AT&T Labs - Research, Florham Park, NJ, 2013

PA2 "An Experimental Evaluation of Multilevel Layout Methods" by Gereon Bartel, Carsten Gutwenger, Karsten Klein, and Petra Mutzel at Technische Universität Dortmund, Germany, 2011

PA3 "Force-Directed Drawing Algorithms" by Stephen G. Kobourov at University of Arizona, 2004

PA4 "Graph Drawing by Stress Majorization" Emden R. Gansner, Yehuda Koren and Stephen North at AT&T Labs — Research, Florham Park, NJ 07932, 2005

PA5 "Visualizing large graphs" by Yifan Hu and Lei Shi at Yahoo Labs, 111 W 40th St, New York, NY 10018, USA. and SKLCS, Institute of Software, Chinese Academy of Sciences, China, 2015

2.5 Overview

This software design document is divided into three sections. The first section gives an overview about the document, while the second and the third section concentrate on the concrete architecture of the current implementation (section 2) and our proposed implementation (section 3).

3 Current Architecture

Our software is not replacing an existing product. However, our software builds on top of VANTED. VANTED is supplying a plugin architecture our product will build on.

4 Proposed Architecture

4.1 Overview

The software that is developed by our team is divided into two independent parts: The Stress Minimization Add-on and the Multilevel Framework Add-on.

4.1.1 Stress Minimization Add-on

The Stress Minimization Add-on shall be an add-on for the software VANTED that is able to layout the graphs provided by VANTED, using a stress minimization algorithm. The add-on shall use the add-on architecture provided by VANTED to add the stress minimization algorithm to the list of selectable algorithms within the software.

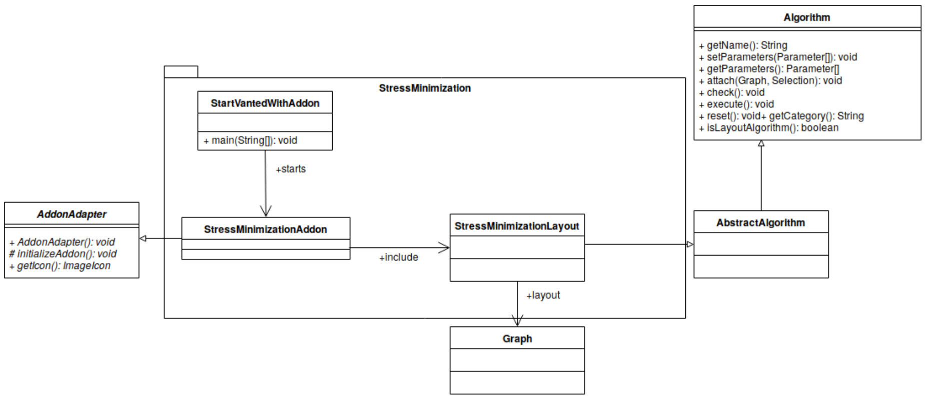

Figure 1: class diagram for the stress minimization add-on

4.1.2 Multilevel Framework Add-on

As the Stress Minimization Add-on the Multilevel Framework Add-on shall integrate a new layout method into VANTED using it's plugin infrastructure. However, this add-on will not supply simply a new algorithm, but a framework for execution existing algorithms within. For this purpose this part of the software will make use of the algorithm management system in VANTED and use GUI components provided either by VANTED or other plugins registered to it.

Figure 2 shows the proposed class architecture for the multilevel framework. Like all add-ons for VANTED there will be a class in our case called MultilevelLayoutAddon extending from AddonAdapter. In this class all algorithms, views and tabs of the add-on will be listed. The central building piece of the framework will be the MultilevelLayoutAlgorithm class. This class inherits from the ThreadSafeAlgorithm class, which is an implementation of the Algorithm interface.
Each instance of MultilevelLayoutAlgorithm will be created with a layout, a coarsening and a placement algorithm as parameters. Layout algorithms are identified with the isLayoutAlgorithm() method from the Algorithm interface. There will be interfaces for the coarsening and placement algorithms in the add-on. This is necessary to allow the user to add further custom made algorithms to our framework through add-ons.
The data model of VANTED alone is not sufficient for the implementation of the multilevel framework. A MultilevelGraph is composed of numbered GraphLevels, which is a subclass of the class Graph provided by VANTED. Each MultilevelNode within these GraphLevels has to reference all of its child nodes in the GraphLevel below.
After the last coarsening step all levels of the graph will have to be present in the memory. However we still expect the required memory space to scale linearly

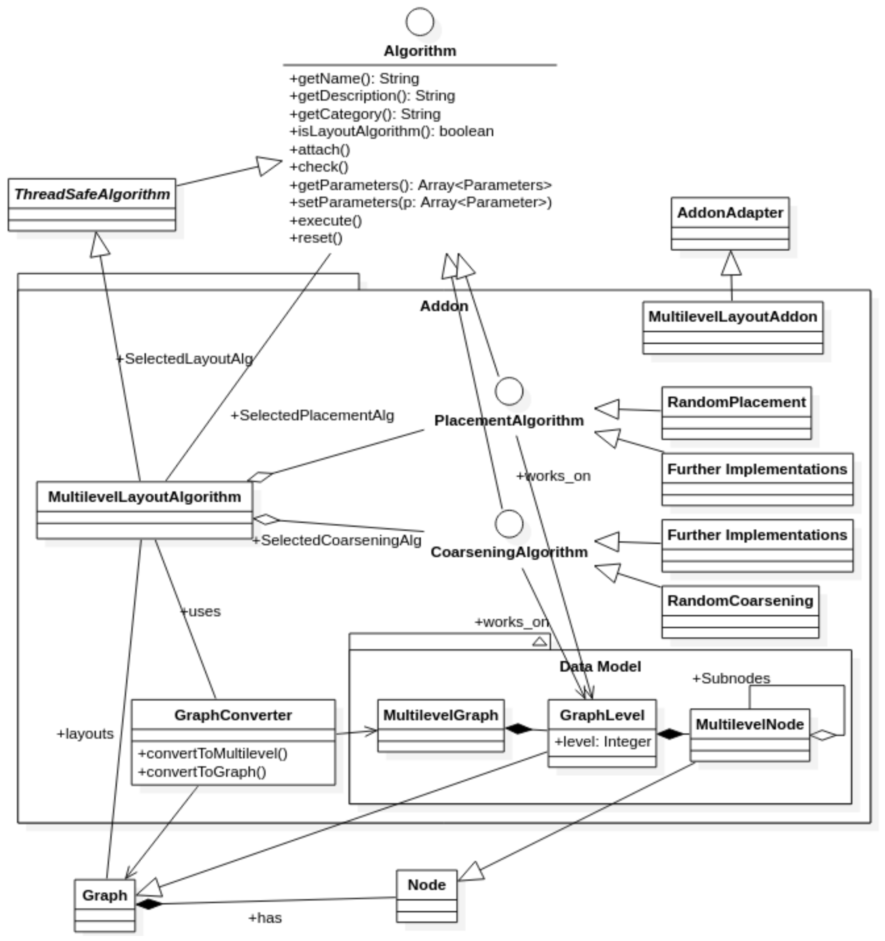

Figure 2: class diagram for the multilevel framework

with an increasing number of nodes in the input graph, since it can reasonably be assumed that for most implementations each coarsening step at least halves the number of nodes and thus the total number of nodes on all levels can be estimated by the geometric series.

4.2 Subsystem Decomposition

Figure 3 describes the subsystem decomposition of our software. As described above, there are two independent Add-ons within the software. The Multilevel Add-on contains two further subcomponents that contain at least two implementations of a coarsening and placement algorithm. These two parts are factored out, since these components of the multilevel framework are replaceable and their implementation is largely independent of the implementation of the

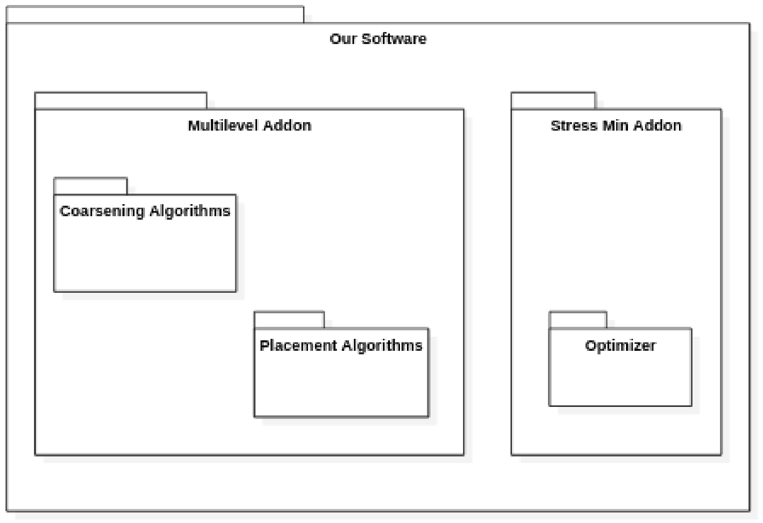

Figure 3: Packages of our software

framework itself. The contents of both packages rely only on the public interfaces provided by the implementation of the multilevel framework itself.
The other Stress Minimization Add-on on the other hand contains a package that performs the optimization process. This process is rather engaged and therefor factored out to keep the implementation clean.

The different components and interfaces of our two add-ons are described in more detail in figure 4. Both components use VANTEDs general add-on interfaces, specifically the Add-on interface and the Algorithm interface, but since background execution is preferred for long calculations required by both algorithms, both plugins also use the background execution services offered by VANTED. The Multilevel Framework Add-on also access the algorithm management unit of VANTED that provides access to the list of algorithms registered to the program.
The Stress Minimization Add-on also makes use of the Interface provided by the optimizer subsystem.
The Multilevel Framework provides new interfaces for coarsening and placement algorithms and the two packages these algorithms are located work on these interfaces.

4.3 Hardware/ Software Mapping

Our software will be assigned to the hardware as in figure 5. The environment

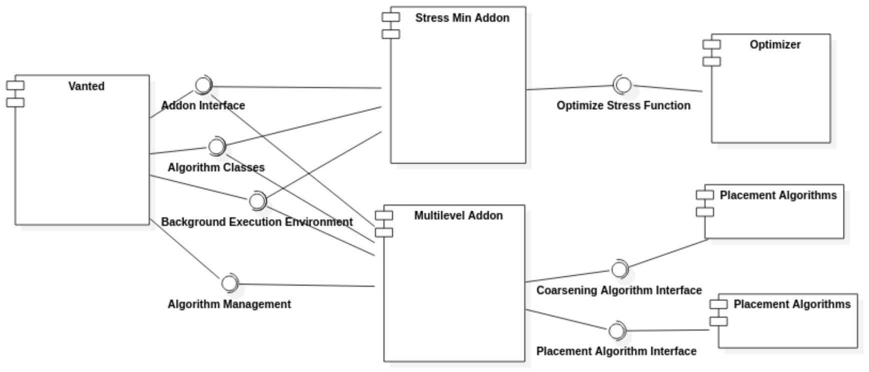

Figure 4: Different Components and Interfaces

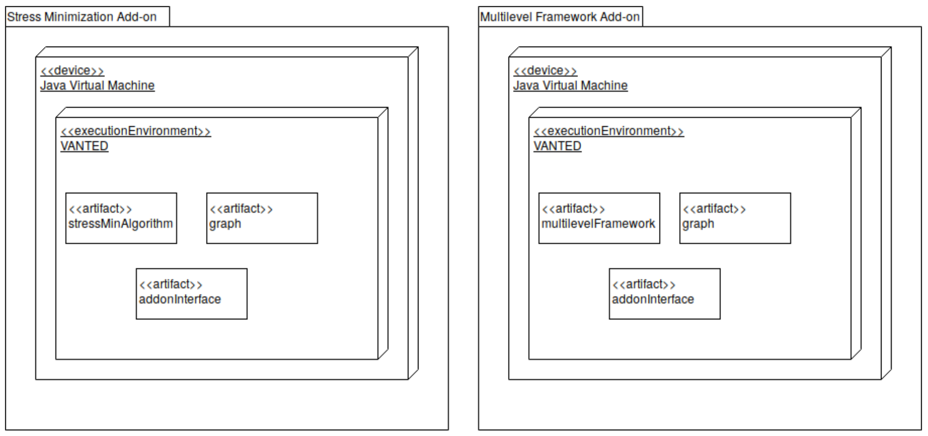

Figure 5: Components of our two add-ons

on the PC is VANTED, the algorithm interface is already part of VANTED. Our stress minimization algorithm takes a graph and transforms it. Graphs are already a part of VANTED as well. The same applies for the Multilevel Framework Add-on.

4.4 Persistent Data Management

All persistent data is managed by VANTED and not part of any of our add-ons.

4.5 Global Software Control

4.5.1 Stress Minimization Add-on

The algorithm is initiated by VANTED with a graph generated by the user in VANTED, as described in figure 6. The algorithm calculates a new layout for the graph and then returns it back to VANTED.

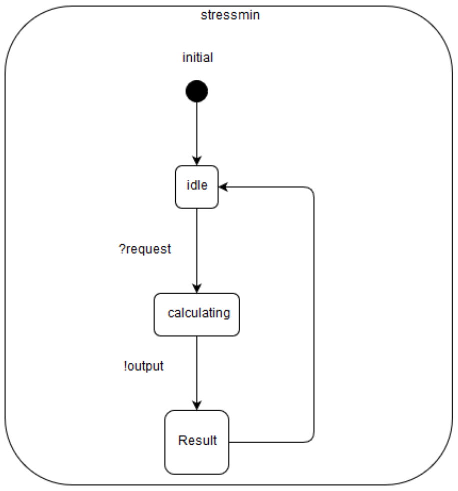

Figure 6: global software control of the stress-minimization-add-on

4.5.2 Multilevel Framework Add-on

In Figure 7, the general execution process of the multilevel framework is summarized. Execution starts when VANTED creates the multilevel algorithm after user interaction. It then executes a number of calls. Only the calls of `check` and `execute` are listed in the diagram, since these methods are especially relevant for algorithm execution. However, calls of `attach` and `getParameters` as well as `setParameters` are also performed. When execution begins, the class that is registered to VANTED as Algorithm class, creates the Coarsening, Layout and Placement Algorithm instances that were specified in the parameters dialogue. After these instances were created, a new thread is started using the

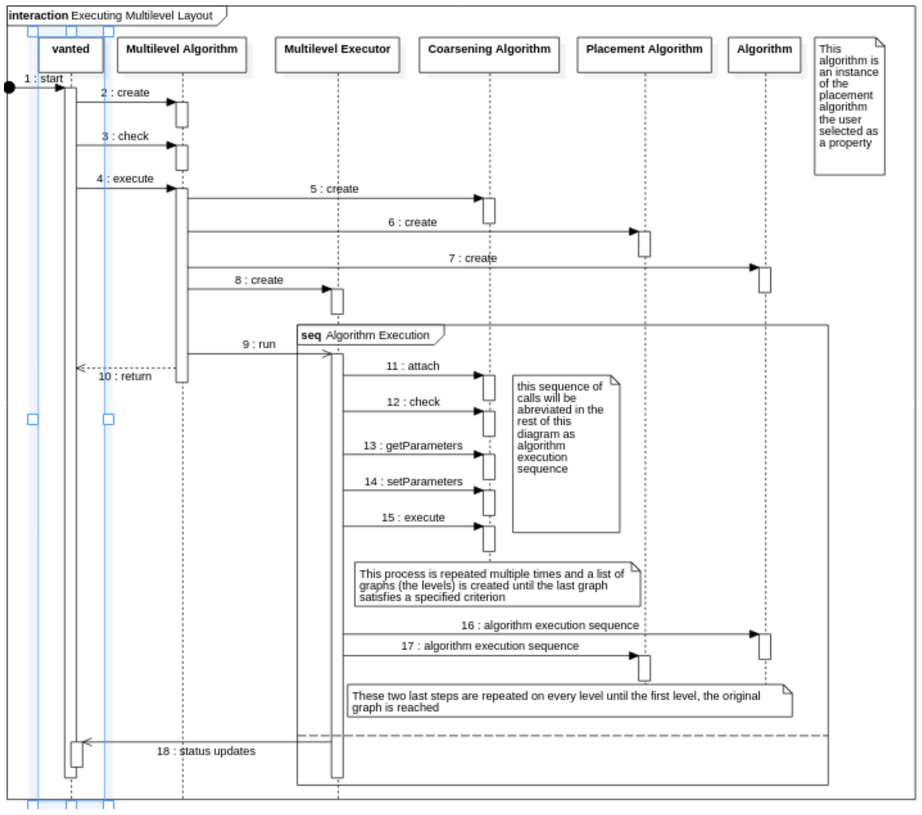
interaction Executing Multilevel Layout
vanted
Multilevel Algorithm
Multilevel Executor
Coarsening Algorithm
Placement Algorithm
Algorithm
This algorithm is an instance of the placement algorithm the user selected as a property
1 : start
2 : create
3 : check
4 : execute
5 : create
6 : create
7 : create
8 : create
seq Algorithm Execution
9 : run
10 : return
11 : attach
12 : check
13 : getParameters
14 : setParameters
15 : execute
this sequence of calls will be abreviated in the rest of this diagram as algorithm execution sequence
This process is repeated multiple times and a list of graphs (the levels) is created until the last graph satisfies a specified criterion
16 : algorithm execution sequence
17 : algorithm execution sequence
These two last steps are repeated on every level until the first level, the original graph is reached
18 : status updates

Figure 7: Execution of the Multilevel Framework.

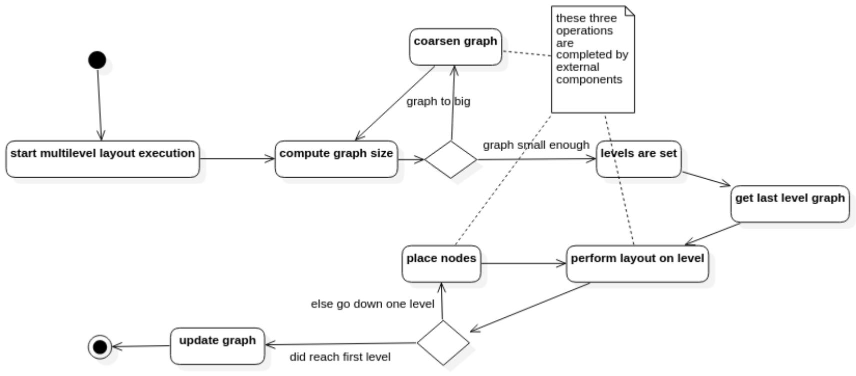

Figure 8: Detailed Execution Steps of the Multilevel Framework.

background services offered by vanted. The sequence of calls performed to start this background activity is abbreviated in the `run` call. The instance that is called Multilevel Executor in the diagram then performs the multilevel framework algorithm itself by calling the coarsening, layout and placement algorithms that were created before. The class needs to go through the whole algorithm call sequence that is specified in VANTEDs documentation. We may not use the service offered by VANTED for this purpose, since these classes do not allow us to set parameters or attach custom graphs. While these algorithms are running, the background activity also reports status updates to VANTED, these status updates include simple progress reports, but also include updates of the active graph as part of an animated algorithm execution.

Figure 8 describes the algorithm execution in more detail. First, the input graph is reduced in the iterative coarsening process. The coarsening itself is done by a separate algorithm, as described above. After this process has finished, on each level the layout algorithm is executed and if the first level was not reached, the aggregated nodes on the level below are placed by the placement algorithm. If the process has reached the first level, execution is finished and the displayed graph is updated. The process gets more involved if the process is being animated, since the displayed graph needs to be updated more often.

4.6 Access Control and Security

No access control or security mechanisms beyond those of the operating system, the java environment and VANTED will be implemented in our software.

4.7 Boundary Conditions

4.7.1 Stress Minimization Add-on

The sequence diagram (Figure 9) describes the interaction between user, VANTED and the stress minimization add-on to starts the stress minimization algorithm, including a failure behavior. First the user starts VANTED and creates a graph. Then there appears a new tab called layout. The user clicks on this tab, selects the stress minimization algorithm and chooses parameters. To start the layout algorithm the user has to click the Layout Network button. Next the stress minimization algorithm checks if the graph is empty. When the graph is empty the stress minimization add-on returns an error message to VANTED. If the graph is not empty and the user checks the checkbox Auto Redraw the stress minimization add-on transform the graph layout and returns the resulting graph layout after each iteration of the algorithm. If the user clicks the Layout Network button without checking the redraw button, the stress algorithm runs and returns the transform graph layout at the end.

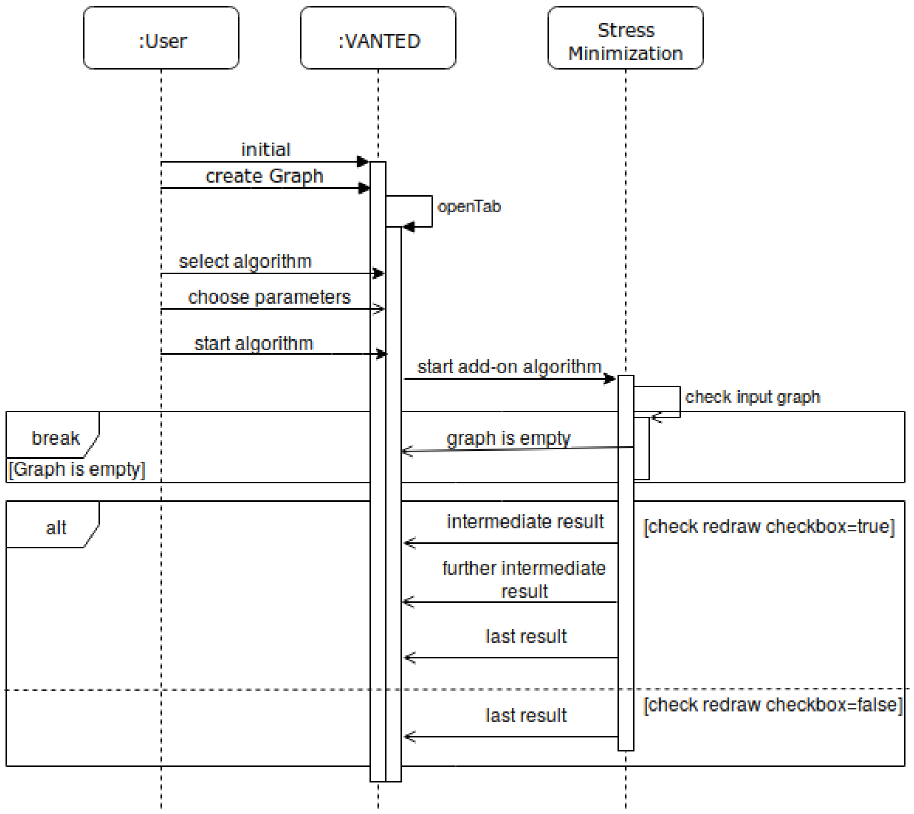

Figure 9: start process of the stress minimization add-on

4.7.2 Multilevel Framework Add-on

For the multilevel framework special attention has to be given to the output of the modular algorithms. For example it is possible that the coarsening algorithm in one step doesn't decrease the size of the graph and the framework enters an infinite loop of coarsening steps because the break condition is never reached.
If an error occurs within one of the used algorithms, it also has to be handled by the framework.
Figure 10 shows an examplary sequence of interactions of an user and VANTED

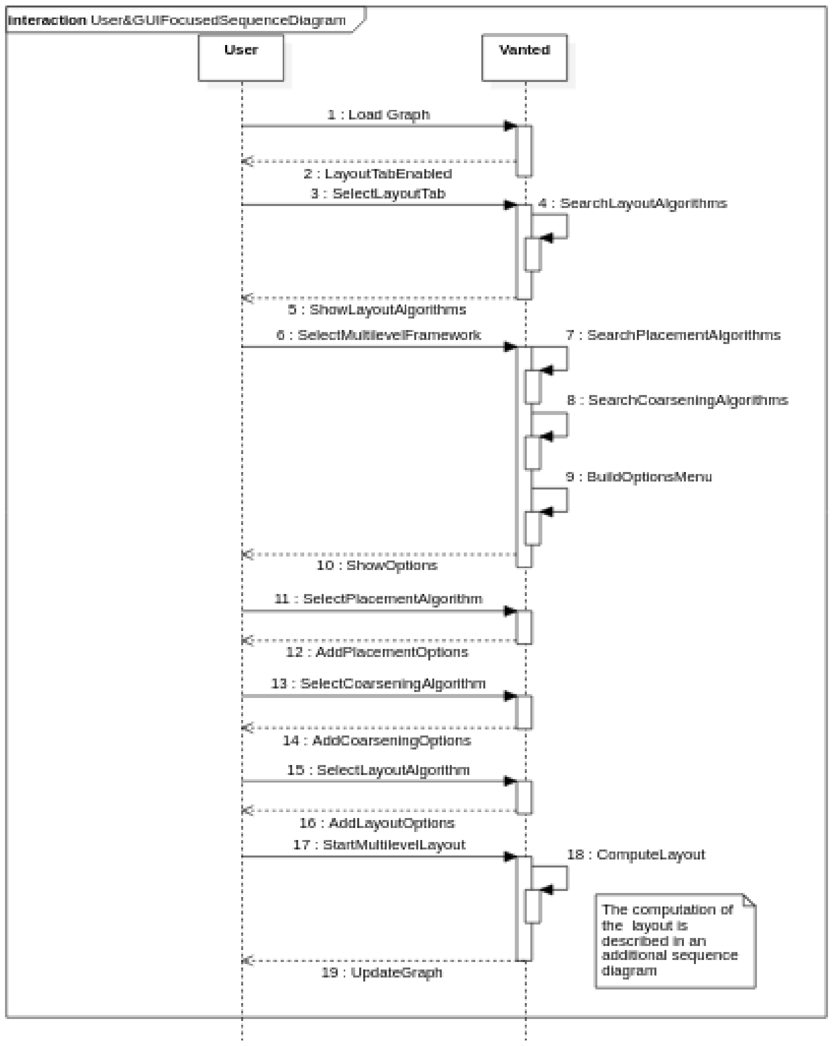
Interaction User&GUIFocusedSequenceDiagram
User
Vanted
1 : Load Graph
2 : LayoutTabEnabled
3 : SelectLayoutTab
4 : SearchLayoutAlgorithms
5 : ShowLayoutAlgorithms
6 : SelectMultilevelFramework
7 : SearchPlacementAlgorithms
8 : SearchCoarseningAlgorithms
9 : BuildOptionsMenu
10 : ShowOptions
11 : SelectPlacementAlgorithm
12 : AddPlacementOptions
13 : SelectCoarseningAlgorithm
14 : AddCoarseningOptions
15 : SelectLayoutAlgorithm
16 : AddLayoutOptions
17 : StartMultilevelLayout
18 : ComputeLayout
The computation of the layout is described in an additional sequence diagram
19 : UpdateGraph

Figure 10: Sequence diagram for typical usage of the multilevel framework

with the multilevel add-on enabled. After the multilevel framework is eventually selected in the layout tab, the user will be presented with lists of algorithms for layout, placement and coarsening algorithms.
After selecting one of each of the three types of algorithms the user will be presented with further options to specify the parameters for the selected algorithms. The computation of the layout is detailed in figure 7.