Software Design Document
Team 2.2

Table of Contents

Introduction

Purpose of the System

The goal is to create two add-ons for the software VANTED (http://vanted.org) that implement fast stress minimization and a multilevel graph drawing framework respectively. The add-ons are to be developed for the class “Softwareprojekt” in the summer term 2019 at the University of Konstanz. This document lays out the requirements for the add-ons.

Design Goals

MLF: Development of an add-on for the VANTED software that provides a multilevel framework. It shall provide the following:

- Performance that is not unreasonably slower than the ogdf::ModularMultilevelMixer implementation.
- Possibility to choose and customize a layout algorithm (via the VANTED graphical interface) that will be applied on every level of the graph.
- Provide an interface for adding other coarsening and placement methods.

SM: Development of an add-on for VANTED that adds a stress minimization layout to it. It shall provide the following:

- The add-on shall be able to work correctly with large graphs.
- It shall not perform unreasonably slower than ogdf::StressMinimization.
- The add-on shall be thoroughly tested and benchmarked.

Definitions, Acronyms and Abbreviations

- **VANTED** refers to “Visualisation and Analysis of Networks containing Experimental Data”.
- **Add-on** refers to an extension of the VANTED software that can be loaded by VANTED and enhances its functionality.
- **OGDF** refers to “Open Graph Drawing Framework” (http://www.ogdf.net/doku.php)
- **GUI** refers to “Graphical User Interface”
- **MLF** refers to “Multilevel Framework”
- **SM** refers to “Stress Minimization”
- **Large Graph** refers to graphs consisting of more than 1000-3000 vertices.

References

- **[Spec]** Softwareprojekt2019_Gruppe2Thema.pdf
- **[V]** VANTED source code (https://bitbucket.org/vanted-dev/vanted/src/master/)
- **[Meet]** The initial meeting on the 25 April 2019
- **[Meet2]** The supervisor meeting on the 2 May 2019
- **[Meet3]** The supervisor meeting on the 16 May 2019

- **[GKN04]** “Graph Drawing by Stress Majorization” (GKN04.pdf)
- **[Eval]** Bartel2011_Chapter_AnExperimentalEvaluationOfMult.pdf
- **[SRS]** Software Requirements Document

Overview

This document specifies how the MLF and the SM-algorithm shall be designed and is categorized into three chapters. The first gives an overview of the contents of the SDD. The second chapter gives a description of the current architecture and the third chapter describes the proposed software architecture for the application.

Current Architecture

In the current state none of the add-ons is implemented. VANTED already has an add-on interface that will be used.

Proposed Software Architecture

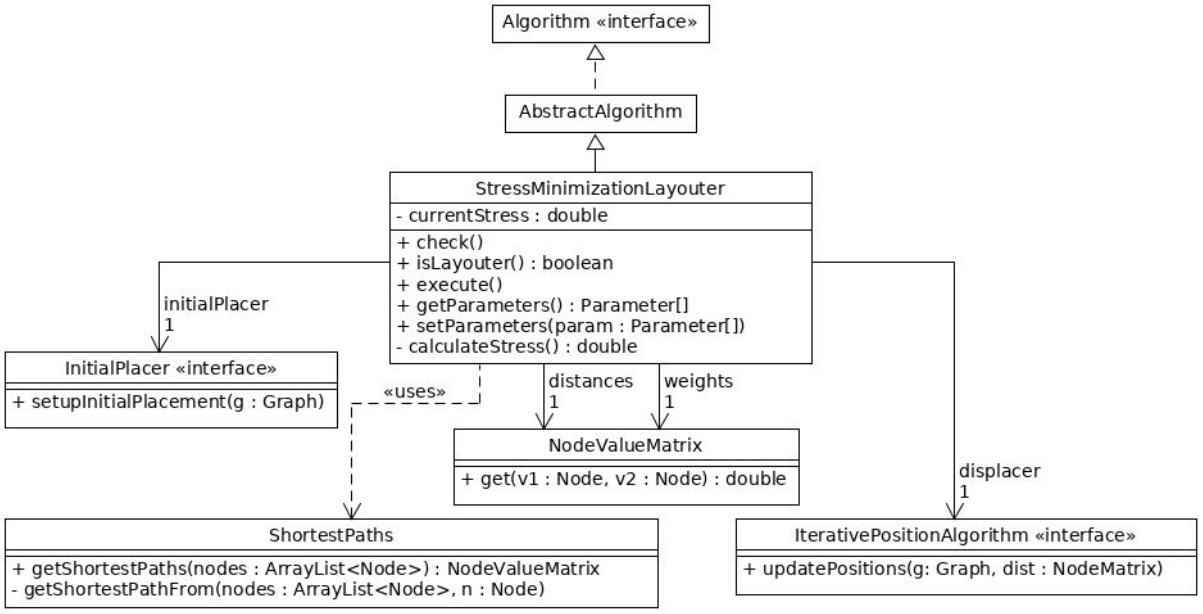
Overview

MLF: Implementation of an add-on for VANTED that adds a multilevel framework layouter. The layouter shall be modular in that it shall be possible to run it with different mergers, placers and level layouters. This shall be achieved by providing an interface that the used mergers and placers must implement. The interaction with the layouters happens through the *Algorithm* interface that already exists within VANTED. A data structure (*MultilevelGraph*) for storing and manipulating the coarsening levels of the graph shall be implemented. It is to be used by the implementers of the *Merger* and the *Placer* interfaces and needs to implement the *Graph* interface so that the level layouters can be used on it.

SM: Implementation of a network layout method for undirected graphs (“Stress Minimization”). A simple iterative variant as a basic version is implemented.
Initially unweighted shortest paths between all node pairs calculated, then an initial layout is determined, and then iteratively improved until a termination criterion is reached. That means if the difference in stress or the positions falls below a certain threshold “ε” or a maximum number of iterations passed (which both can be set by the user).

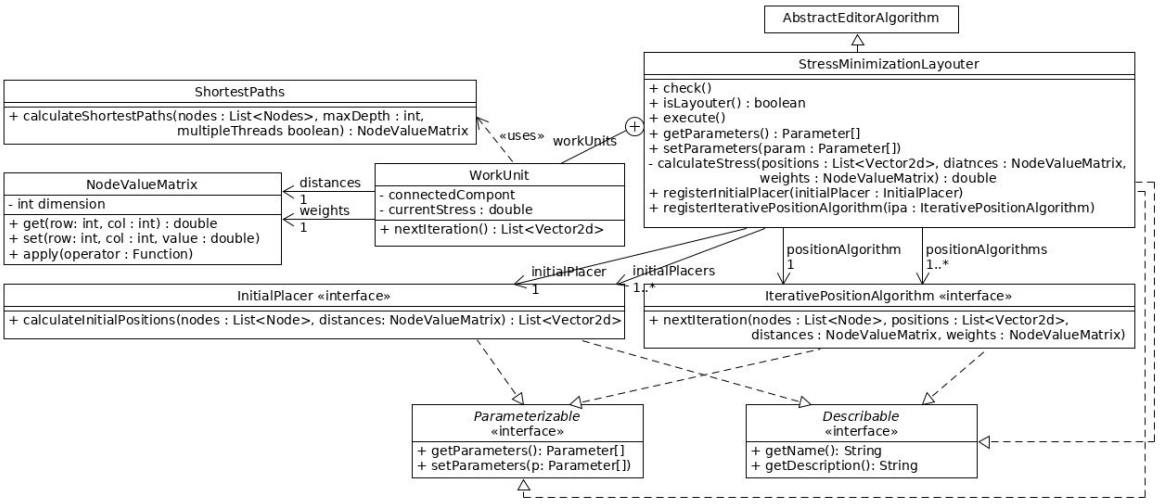
Class Diagram

Stress Minimization:



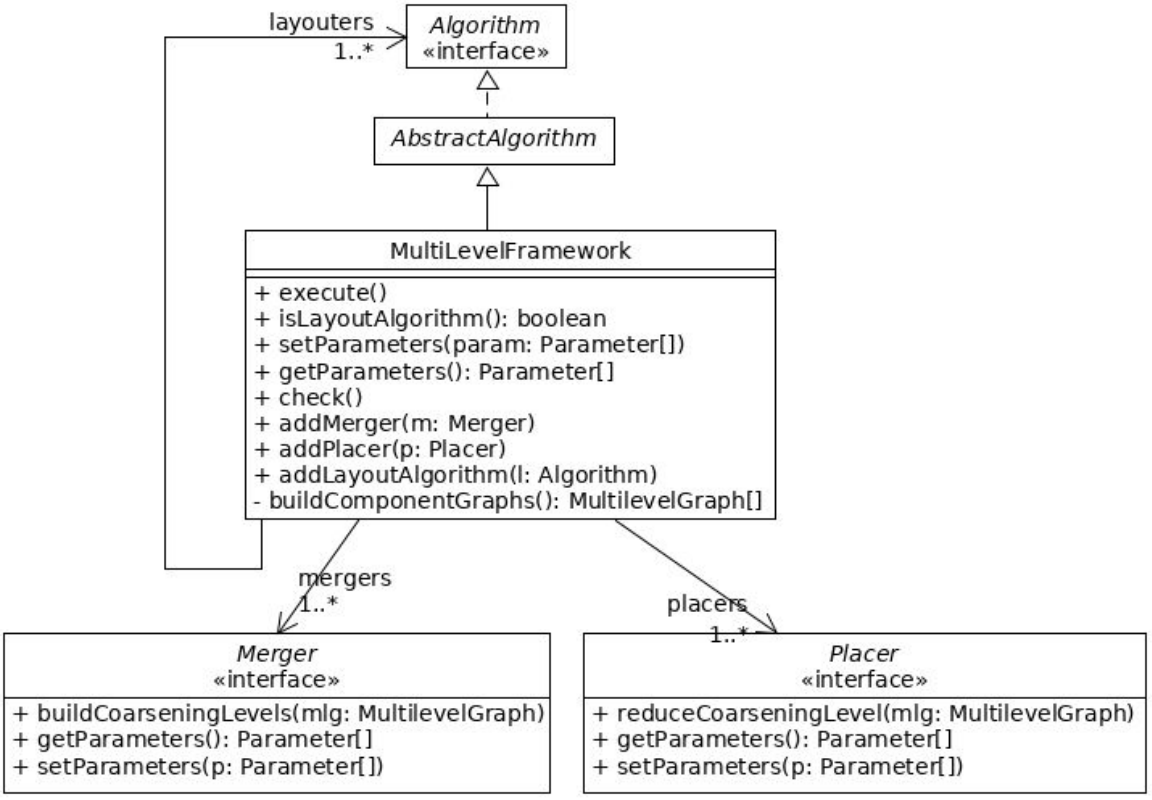
The main class of the SM-algorithm is the *StressMinimizationLayouter* class. It has multiple helper classes. *ShortestPaths* will be used first to calculate the graph theoretical distances between the vertices. The “InitialPlacer” is used to determine an advantageous initial layout. After the preprocessing phase *IterativePositionAlgorithm* and “calculateStress()” of the main class are used to iteratively to get a better layout until the desired threshold or max iterations are reached.
NodeValueMatrix is a class that associates a value for every pair of nodes. This is used internally to store the node distances and weights.
IterativePositionAlgorithm and *InitalPlacer* are only interfaces to enable the implementation of different methods with varying degrees of effectiveness.

Updated Version



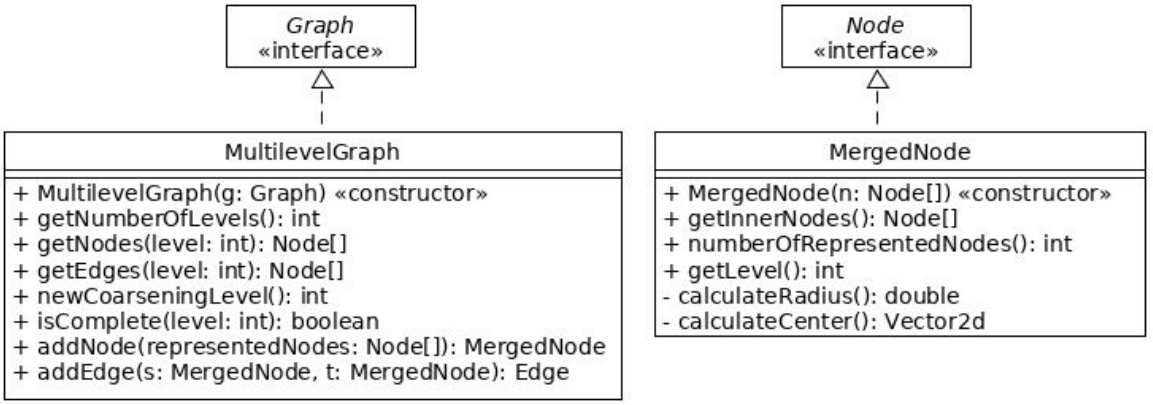
The general design is the same but it got more specific and some methods have been renamed. Now an inner class called “WorkUnit” that is created for every connected component and does the actual calculations. Every work unit can work on its connected component and finish the algorithm independently.
The InitialPlacers and IterativePositionAlgorithms are now displayable by the GUI and are required to have a name and description because of that. They also provide parameters like the main algorithm. Both functionalities have been extracted into an interface.
It is now possible to register more InitialPlacers and IterativePositionAlgorithms using a static method in StressMinimizationLayout. (Other new classes have been omitted for clarity.)

Multilevel Framework



The MLF shown above implements the *AbstractAlgorithm* interface so it can be executed by the users of VANTED. The user can choose and set up a *Merger,* a *Placer* and a *Layouter* from given options. On execution these are used to calculate a layout iteratively. First the *Merger* builds a *MultilevelGraph* containing a limited amount of levels. Afterwards starting with the top level intermediate layouts are computed. The *Placer* uses these layouts to place nodes in the lower levels of the multilevel graph. The MLF contains the method *buildComponentGraphs* which is used to split disconnected graphs into their connected components so they can be layouted separately.

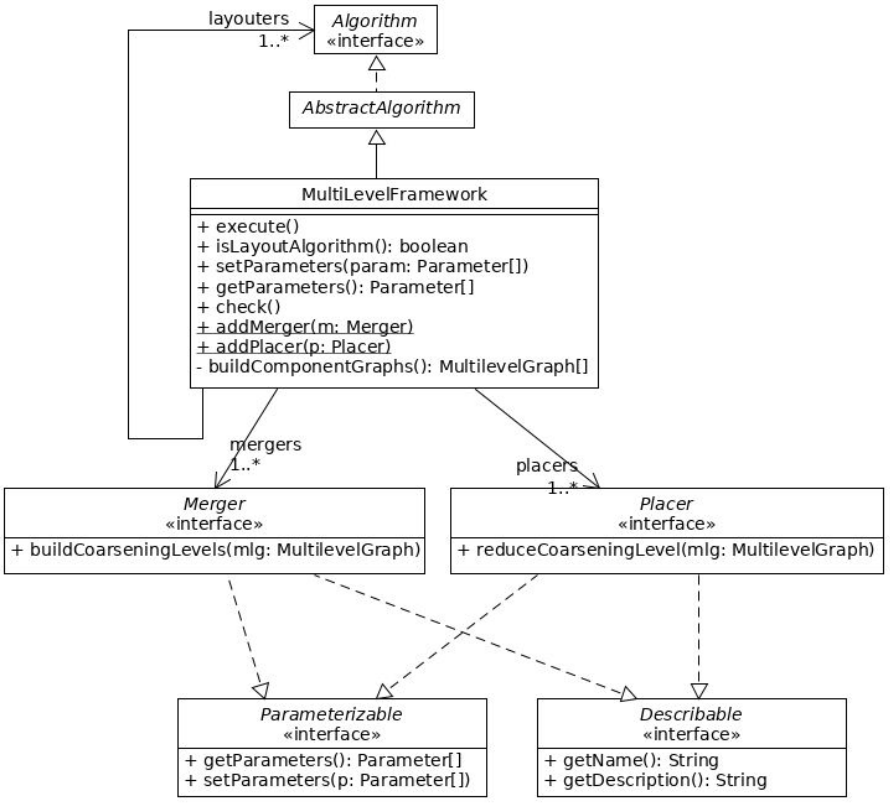
The data structures used by the framework are shown in the following class diagram:



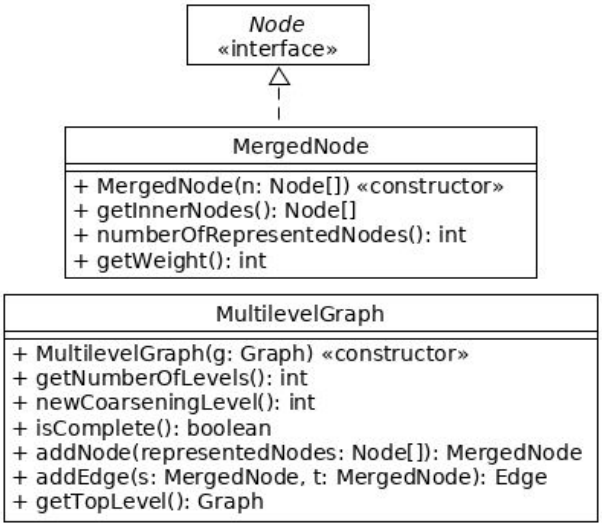
MergedNode implements the *Node* interface of VANTED. It stores a node of a coarsened graph that represents multiple nodes of the lower level. The *MergedNode* can calculate a radius that is big enough to contain all the lower levels nodes it represents. *MultilevelGraph* implements the VANTED *Graph* interface such that a layouter can be executed on it. It contains all graphs for the levels created by the Merger. The *addNode* and *addEdge* methods operate on the current (i.e. highest) level of coarsening. The *isComplete* method checks if all the nodes from the lower level are represented in the current level.

Updated Version

The general design is the same, but some details were changed in the implementation phase and some classes were added. The updated versions of the class diagrams above are shown below (new classes omitted for brevity).



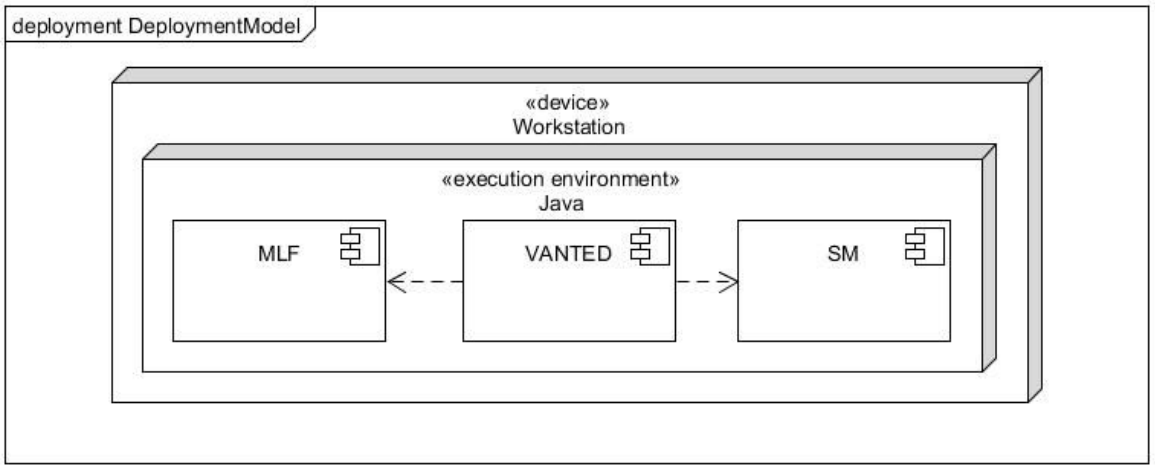
The `addMerger()` and `addPlacer()` methods are now static. The parameter support has been factored out into an interface. `Merger`s and `Placer`s also have to implement the `Describable` interface so they can provide names and descriptions to be displayed in the GUI.



`MultilevelGraph` no longer extends `Graph`. It has a method to return the top level graph to compensate for this. `MergedNode` now has the ability to calculate the weight (the number of nodes of the original graph that the `MergedNode` represents).

Hardware/Software Mapping

In this section, it is described how the subsystems are assigned to the related hardware.



Both add-ons run on the Java Virtual Machine as extensions for the VANTED software. They provide the framework and stress minimization functionality respectively to VANTED. These functions can be called from VANTED. Thus the user interface is also provided by VANTED.

Persistent Data Management

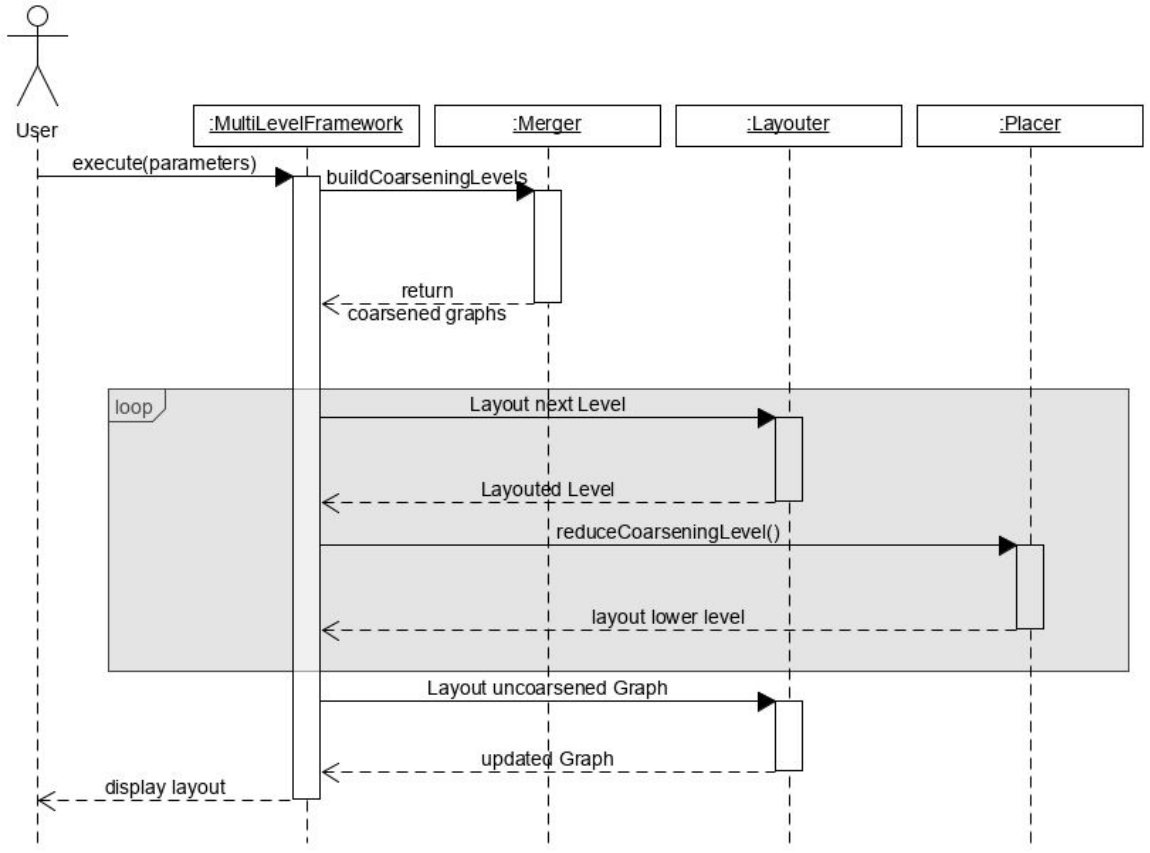
Data Management is controlled by VANTED, raw graph data and layouted graphs can be saved and opened as for example CSV or GML format.

Global Software Control

The software gets controlled by the VANTED-GUI. The add-ons provide a set of parameters to VANTED via an interface. VANTED then displays these parameters (depending on the type of data) as e.g. sliders to the users. They can make some adjustments. When the framework or the algorithm is run they can read the possibly changed parameters and adjust their behaviour accordingly. As a part of VANTED a user can stop any running algorithm or the framework at any time. If this happens the stress minimization shall return the layout of the graph at the current iteration step whereas the multilevel framework shall return the graph to its original state.

Interaction of subsystems and initiation of requests

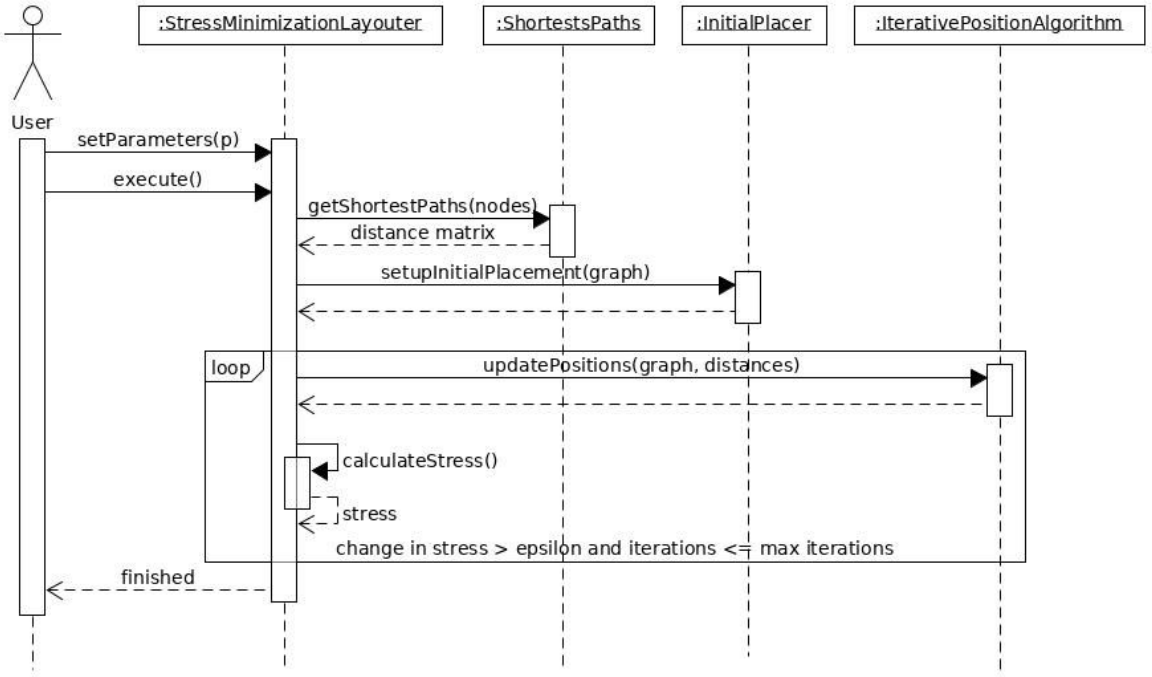
Multilevel framework



The user can set up parameters for the MLF and the layouter and start the layouting process. The MLF then proceeds to build the coarsening levels. The parameters that the user set up for the layouter are passed to it. Then the layout of the levels is carried out in a loop for the different levels, starting with the coarsest one. After each level layout (except for the last one) the nodes from the previous level are placed back into the graph using the placer. At last, the layouter is run on the original graph.

The user can also decide to abort the multilevel framework by clicking the respective button in VANTED (not shown in the sequence diagram). This will restore the original graph layout.

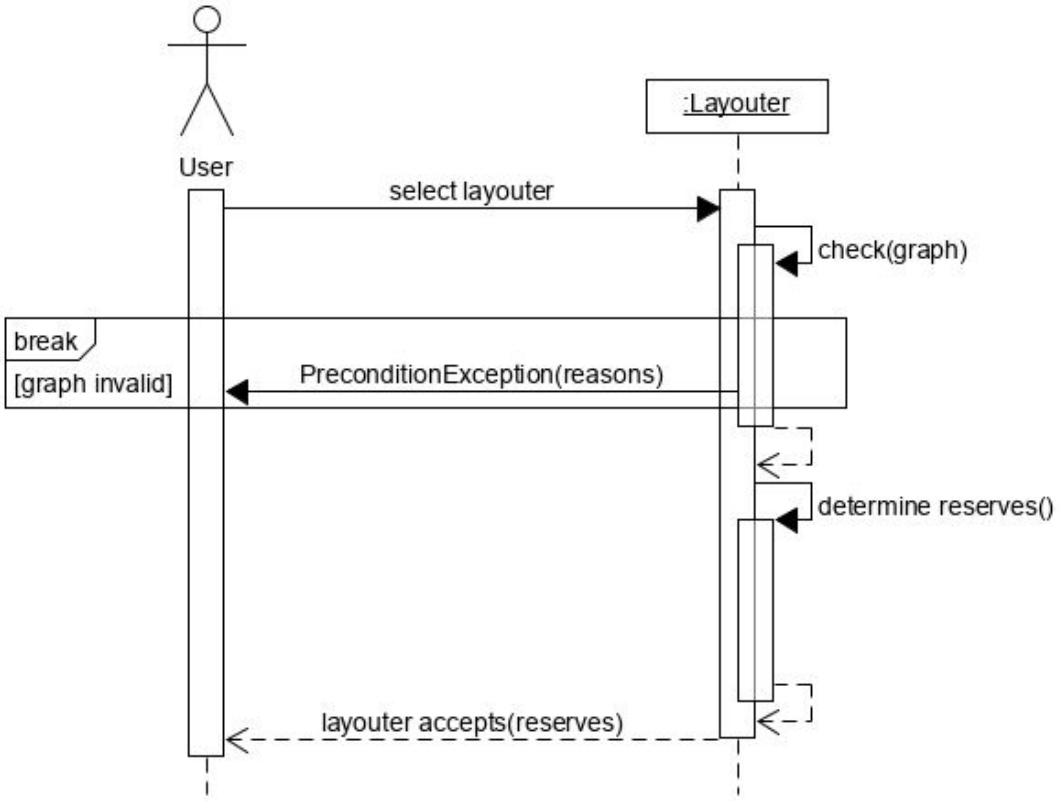
Stress minimization



First the user uses the VANTED GUI to select the algorithm and change its parameters. After being started by the user the algorithm first calls *ShortestPaths* to calculate the shortest distance between all vertices. *InitialPlacer* is used to determine an advantageous initial layout and the weights are calculated from the distances. Then it tries to move the vertices to improve the layout by calculating the new positions using the *IterativePositionAlgorithm* class which updates the graph accordingly. This procedure is repeated until the change in the stress function or the difference in total node positions before the vertices were moved and after it are smaller than a user given threshold called “ε”. If this threshold is reached or the number of iterations goes beyond a given maximum the algorithm stops letting the current layout be the final one. If the layouter is stopped by the user, it just returns without resetting the graph. Thus its possible to stop the layouter if the result is good enough and get an immediate result.

Boundary conditions

Start-up and shutdown are managed by the VANTED software and a standard use case with Start-Up and Shut-Down for both add-ons is described in the sequence diagrams above But there are some possible errors. These cases are described below.



The user selects the stress minimization algorithm or the multilevel framework in the layout list. In this case VANTED calls the “check” method of the layouter. The layouter (shown in the diagram) checks whether the current graph meets the required preconditions i.e. is valid. In our case that means e.g. whether is empty; The graph being empty results in a break. No further computations are made until a nonempty graph is omitted. In addition to that the layouter checks the graph for further properties which require specific treatment (e.g. weighted graphs or graphs containing multiple disconnected connected components). If an omitted graph is not fully connected the components of the graph are handled separately. Their respective layouts are shown displayed next to each other. Directed Graphs are treated as well as their edges were not directed and loops are ignored. The stress minimization algorithm additionally ignores weighted edges.

Design decisions

A lot of functionality of VANTED has been reimplemented (like ‘`getConnectedComponents()`’ and ‘`layoutConnectedComponets()`’) because the functions provided by VANTED do not support only working with the current selection. We decided against wrapping these functions to improve the performance.

Stress Minimization

To improve performance we decided to not work with the nodes directly and use `ArrayList`s with a set index for every node. We also decided to work only with the positions of the nodes on the lower levels, to avoid the costly call of ‘`getAttribute()`’ on every node every time the position is needed.
We decided against using an already existing implementation of PivotMDS or its used functions because there does not seem to be an implementation of them in the libraries already included in VANTED and we did not want to add any more dependencies.
New `InitialPlacer`s and `IterativePositionAlgorithm`s can be added using the static methods
‘`registerInitialPlacer(InitialPlacer)`’ and
‘`registerIterativePositionAlgorithm(IterativePositionAlgorithm)`’. Because we wanted to ensure that the user cannot change the parameters while the algorithm is run, a new instance of them is created using an expected default constructor and their parameters are copied using ‘`setParameters(Parameter[])`’.

Multilevel Framework

Some algorithms cannot be used as level layouters within the MLF. Therefore we decided to only use a whitelist of algorithms that the user can choose from.
The whitelist is a member of the `LayoutAlgorithmWrapper` class (`org.vanted.addons.multilevelframework.LayoutAlgorithmWrapper.WHITELIST`). If you want to add an algorithm to the whitelist, make sure it meets the following properties:

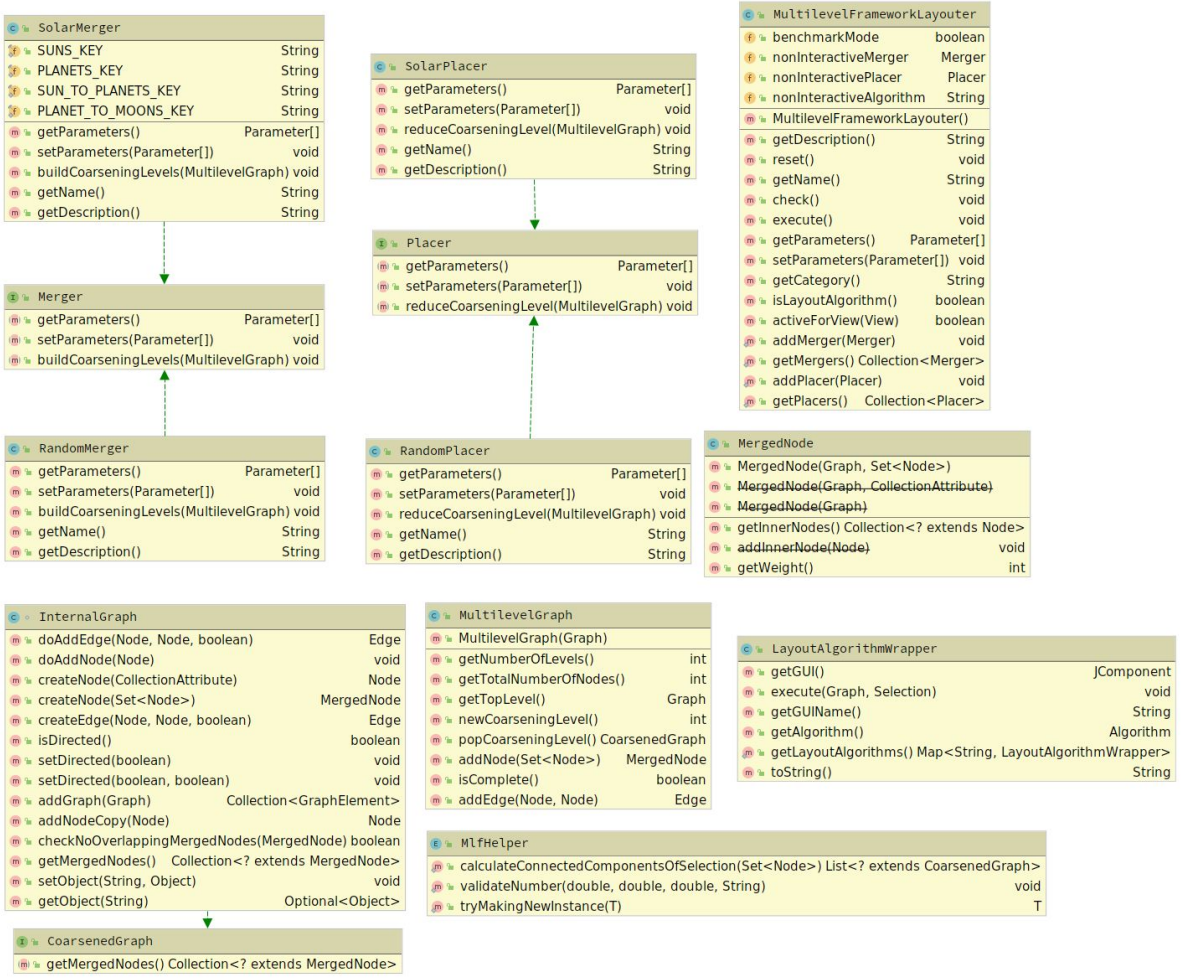
- The `execute()` method terminates only after the algorithm is done executing. Lots of algorithms start a `BackgroundTask` or a `Thread` in the `execute()` method, making it hard for the MLF to detect when the algorithm is done. If you want to start a `BackgroundTask` in the interactive version of your algorithm, but block when the algorithm is run inside the MLF, you can use the following graph attributes to determine if the graph that you are laying out is a level in a `MultilevelGraph`:
    - `GRAPH_IS_MLF_COARSENING_LEVEL`
        - Set to `true` if the graph is a run within the MLF.
    - `GRAPH_IS_MLF_COARSENING_TOP_LEVEL`
        - Set to `true` if the graph is the top level (coarsest level) of a `MultilevelGraph`.
        - If your algorithm has a feature to generate an initial starting layout, you should apply it to the top level and *only* to the top level.
    - `GRAPH_IS_MLF_COARSENING_BOTTOM_LEVEL`

    - Set to `true` if the current coarsening level is a connected component of the original graph.
    - The SM add-on uses this functionality to change the edge scaling on upper levels. (It scales relative to the node size and the `MergedNode`s in upper levels can get quite big if they contain lots of nodes, so a different scaling factor is used on upper levels).
- The algorithm does not require the graph to be rendered (in a `View`). This is why the “Adaptagrams Edge Routing” algorithm does not work within the MLF (its internal method `getNodeShape()` requires a `View`). Not that we did develop support for displaying the levels during the layout. It is commented out, however, since it would probably confuse and annoy users. If you want it, remove the comment signs before the two calls to `display()` within the method `org.vanted.addons.multilevelframework.MultilevelFrameworkLayouter.execute()`.

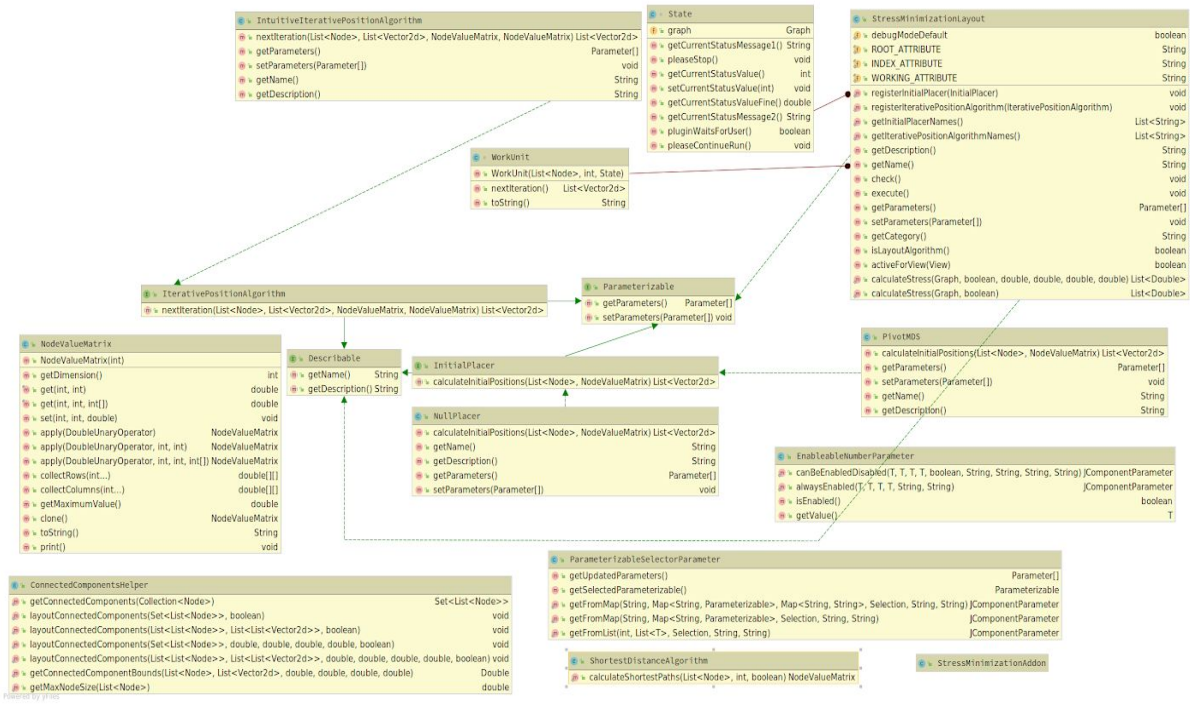
The internal ‘Force Directed’ implementation always runs in a background task and can thus not be observed and waited for by the MLF. Because of that, a custom altered version has to be created that is controllable by the MLF.

Complete Class Diagrams

The following class diagrams were generated using the IntelliJ-IDE.
MLF:



SM:



Changelog

- Added updated version of the UML diagrams to reflect the current implementation.
- Added “Design Decisions” section
- Fixed wrong format of SM UML class diagrams and missing descriptions