Software Requirements Document: Team 2.1

Contents

Introduction

Purpose

The purpose of our software is to speed up the visualization process of large graphs (more than 1000 nodes and edges) and enhance the quality of such visualizations results.
For this purpose we aim at providing two add-ons for the VANTED software.

Scope

The first add-on will integrate the multilayer approach into VANTED that tries to reduce the size of a large graph step wise and uses layout information of the compressed graphs to speed up the overall process, while second add-on will provide a stress minimization algorithm as a further layout algorithm choice.

Definitions, Acronyms, Abbreviations

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

References

INT Introductory lecture on 16/04.

SUB Document "Softwareprojekt2019_Gruppe2Thema.pdf"

ME1 Supervisor meeting 1

ME2 Supervisor meeting 2

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

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

Overview

The SRS is divided into four chapters. The first chapter gives a quick overview about the project and provides the reader with the necessary Definitions, Acronyms and Abbreviations to understand the rest of the document.
The second chapter gives an Overview about the add-ons functions, characteristics and constraints surrounding the project.
The third Chapter contains three diagrams: A use case diagram, an sequence diagram and a class diagram. The main part of that chapter focuses on the requirements of our project, which are written according to IEEE-standards.
The fourth and final chapter contains the Project time schedule, containing our 3 milestones and the project Deadlines.

Overall Description

Product Perspective

The customer wants an implementation of two graph layouts (stress-minimization and multilevel framework) for large networks as an add-on for the java framework VANTED.

Product Functions

The first network layout stress-minimization shall transform large graphs with a stress minimization algorithm into a clearly arranged shape. The add-on adds the algorithm to the selection of available methods in VANTED.

The second add-on is the multilevel framework. The framework works in two steps: in the first step the graph is reduced into a hierarchy of smaller graphs by merging nodes until a minimal graph is reached. In the second step a layout algorithm is applied to the minimal graph and the merging of nodes is undone. This is repeated on every level of the hierarchy until the original size of the graph is reached. This method reduces complexity of operations on every level of the hierarchy and produces better results than just applying a layout algorithm.

User Characteristics

Users of the add-ons are VANTED users. They are familiar with graph theory and have balanced computer skills. VANTED users are mainly situated within life sciences.

Constraints

The add-ons are plugins for the java-software VANTED. The add-ons can not be used independently but only in conjunction with VANTED.

Assumptions and Dependencies

There are tools to create graphs and apply layout algorithm already implemented in VANTED, but there is no method using a stress-minimization approach and no framework for multi-leveling methods of large graphs.

Apportioning Of Requirements

The add-ons do not offer any functions to create a graph. VANTED already offers this function itself. The add-ons return optimal layout for the selected algorithm, but not necessarily the optimal layout for every graph.

Specific Requirements

External Interfaces

Software use cases

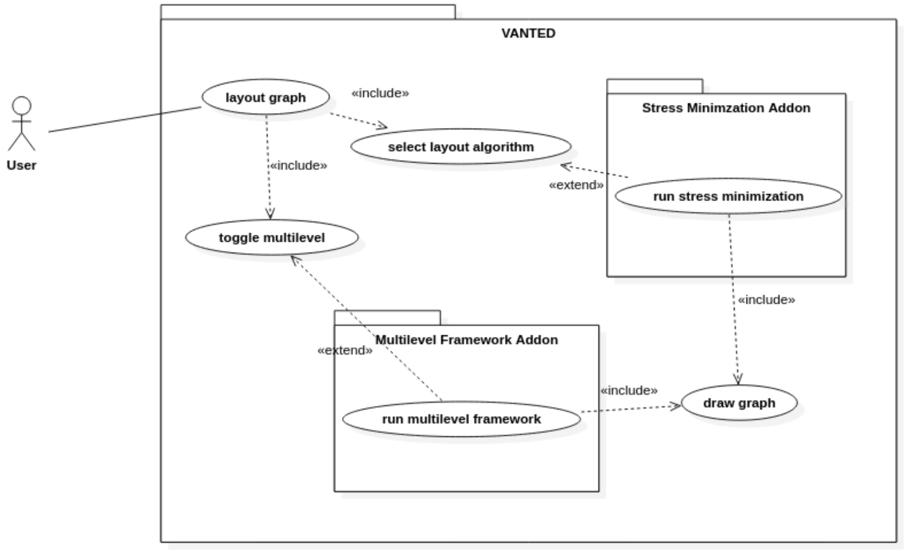

Figure 1: Use Cases of our Software

The use case diagram in figure 1 describes the usage of our software within VANTED. When layouting an algorithm she needs to select a layout algorithm. This algorithm may be the stress minimization algorithm supplied by our add-on. If she choose this algorithm, it is run and the outputs are drawn in VANTEDs graphical interface. Additionally to selecting a layout algorithm the user may choose to enable multilevel execution of the algorithm using the multilevel framework implementation supplied by our software. If she turns on this multilevel option, the algorithm is run multilevel and the outputs are as above shown in VANTEDs user interface.

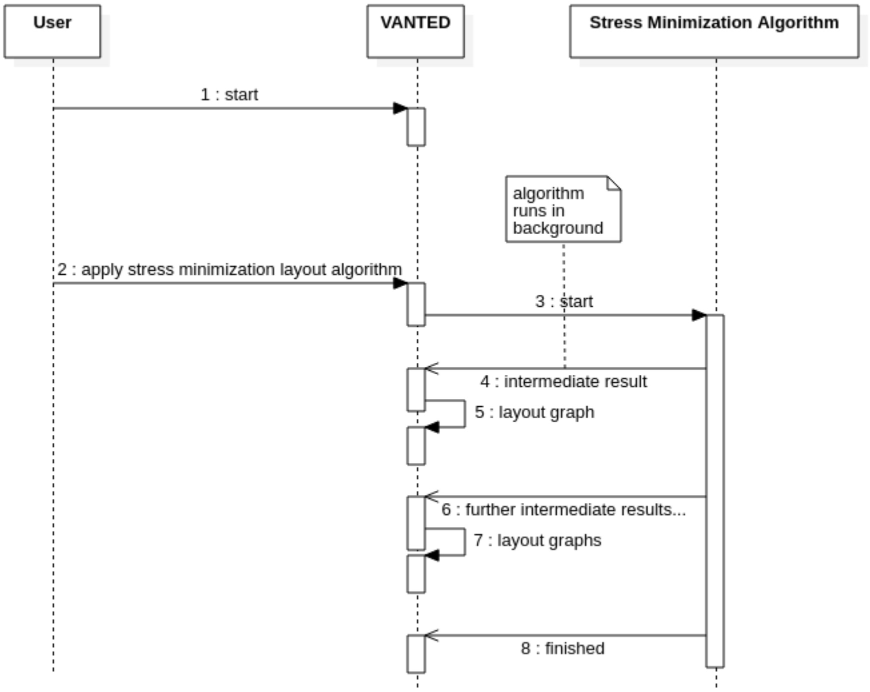

Figure 2: Interactions of the stress minimization layout algorithm with the surrounding Software (VANTED) and asynchronous operations

Interactions of the stress minimization layout algorithm with the surrounding Software (VANTED) and asynchronous operations

The sequence diagram in figure 2 describes the interaction between the user, VANTED and the stress minimization algorithm. First the user starts VANTED and applies the stress minimization layout algorithm to VANTED. VANTED starts the stress minimization algorithm and gets back a intermediate result, while the algorithm is running in the background. Then VANTED layouts the graph, gets further intermediate results and layouts the graphs until the stress minimization algorithm is finished.

Interactions, relationships and dependencies of different entities in context of our software

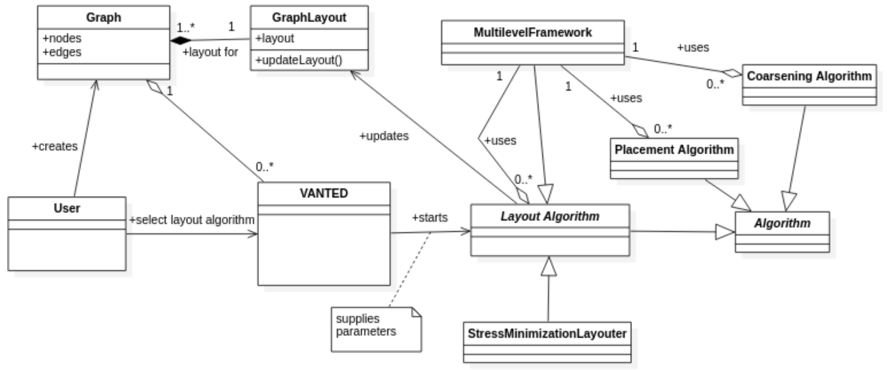

Figure 3: Interactions, relationships and dependencies of different entities in context of our software

The class diagram in figure 3 describes the interactions and relations between different entities of our VANTED add-ons. There are three main types of algorithms relevant for our add-ons: placement, coarsening and layout algorithms. Both the MultilevelFramework as well as the StressMinimizationLayouter are derived from the abstract LayoutAlgorithm class. To function an instance of the MultilevelFramework needs an instance each of a layout, placement and coarsening algorithm. The VANTED user creates graphs and selects layout algorithms to start. LayoutAlgorithms then update the graph layout of the input graph.

Functions

R1: MultilevelFrameworkImplementation
Function: There shall be an implementation of the multilevel framework
Description: The user shall be able to execute any suitable layout algorithm registered to VANTED in a multilevel variant.
Source: SUB, ME1
Dependency: -

R2: MultipleCoarseningAlgorithms
Function: Multiple different coarsening algorithms shall be implemented and integrated into the software for use with the multilevel framework
Description: Since different coarsening algorithms are suitable are most suitable for different graphs and result in different run-times and layouts, the user should be able to choose between at least two such algorithms.
Source: ME2, EM1
Dependency: MultilevelFrameworkImplementation

R3: MultiplePlacementAlgorithms
Function: Multiple different placement algorithms shall be implemented and integrated into the software for use with the multilevel framework
Description: Since different placement algorithms are most suitable for different graphs and result in different run-times and layouts, the user should be able to choose between at least two algorithms.
Source: ME2, EM1
Dependency: MultilevelFrameworkImplementation

R4: AccessToMultilevelFramework
Function: Running the multilevel framework with a selected algorithm shall be very simple and straight forward.
Description: It should be very easy for users to choose the multilevel execution option, for example by activating a checkbox. Further options however also need to be available (See above).
Source: ME1
Dependency: MultilevelFrameworkImplementation

R5: StressMinimizationLayoutImplementation
Function: There shall be an implementation of the stress minimization layout algorithm
Description: Users should be enabled to automatically perform a stress

minimization layout algorithm within VANTED
Source: SUB, ME1
Dependency: -

R6: PresentationAsOtherLayoutAlgorithms
Function: The stress minimization algorithms presentation shall fit into VANTEDs GUI
Description: The stress minimization layout algorithm shall be presented in a very similar way other present layout algorithms are presented.
Source: SUB, ME1
Dependency: StressMinimizationLayoutImplementation

R7: StressMinimizationProgressAnimation
Function: The stress minimization algorithm shall output intermediate results for graphical output.
Description: When running the stress minimization layout algorithm, intermediate steps of the algorithm in form of a real-time layout progress animation shall be presented to the user, if such an option was chosen.
Source: ME1
Dependency: StressMinimizationLayoutImplementation

R8: StoppingAlgorithms
Function: Algorithms shall be able to be stopped during execution.
Description: The user shall be able to stop the execution of both the stress minimization algorithm and the multilevel framework.
Source: IM1
Dependency: StressMinimizationLayoutImplementation, MultilevelFrameworkImplementation

Performance Requirements

R9: ImplementationPerformance
Function: Implementations shall have optimal known time and space complexity
Description: The implementations of the multilevel framework and the stress minimization layout algorithm shall lie within the asymptotic complexity class that are discussed in the papers PA1-5.
Source: ME1
Dependency: MultilevelFrameworkImplementation, MultipleCoars-

eningAlgorithms, MultiplePlacementAlgorithms, StressMinimizationLayoutImplementation

Logical Database Requirements

No logical database requirements.

Design Constraints

R10: ProjectSchedule
Function: The software shall be developed according the the specified time schedule
Description: Software and software artefacts shall be developed and delivered according to the time schedule at the end of this document.
Source: SUB, ME1
Dependency: -

R11: SeperateVANTEDadd-ons
Function: The core software shall consist of two VANTED add-ons
Description: The functionality of the software needs to be delivered through VANTED add-ons. The multilevel framework and the stress minimization algorithm shall be located in seperate add-ons.
Source: IM1
Dependency: -

R12: OpenSourceSoftware
Function: Our software shall be made available under an open source license.
Description: As VANTED itself, our add-ons shall be licensed as open source software. Please also see section additional comments.
Source: INT
Dependency: -

Software System Attributes

- Maintainability:

  R13: CleanDocumentation
  Function: There shall be documentation available for every class

and method
Description: Every class and every method within the software should have own inline code documentation. Code itself should also be documented with comments.
Source: SUB, ME1
Dependency: -

- Usability:

R14: GettingStartedDocument
Function: There shall be a getting started guide
Description: Users of VANTED should be guided through installtation and usage of the software by a getting started guide
Source: SUB, ME1
Dependency: -

R15: UserFriendlyGUI
Function: The grafical interfaces of our software should integrate well into VANTEDs interface.
Description: Users of VANTED should feel familiar with our addons interfaces and be able to use our software straight forward without having to get used to new interface concepts.
Source: SUB
Dependency: -

- Correctness:

R16: ThoroughlyTestedImplementations
Function: Implementations shall be well tested
Description: All algorithms and other implementations should undergo extensive testing, depending on their complexity.
Source: SUB, ME1
Dependency: -

- Extensiblility:

R17: CustomMultilevelSupportingAlgorithms
Function: Users shall be able to create custom coarsening and placement algorithm that can be used with our multilevel framework add-on.
Description: It should be possible to extend our multilevel framework with new coarsening and placement algorithms using further VANTED add-ons.

Source: ME2
Dependency: MultilevelFrameworkImplementation

Organizing The Specific Requirements

The requirements may be divided into general requirements, requirements regarding the implementation of the multilevel framework and the implementation of the stress minimization algorithm.

Additional Comments

Depending on the concrete licenses of VANTED and other software libraries, we will chose a license for our software. We will take special care the licenses of the used libraries do work together.

To furthermore ensure quality and maintainability, it is desired to use well tested and documented external open source libraries in our software wherever possible.

Project Time Schedule

Week/Final Date	Event/Task
25.4. - 5.5.	Finish Software Requirements Document including Diagrams
5.5	**Personal Deadline:** First version of SRS, 1. Two-Week-Report: David
6.5. - 12.5.	Every Team member introduces himself to VANTED development, creates "Hello World"-Plugin to ensure VANTED is working and the structure is familiar and gets an overview of the scientific papers.
9.5.	**Deadline:** Software Requirements Document
13.5. - 19.5.	Team decides upon design and writes SDD
19.5.	**Milestone 1**: Software Design Document is finished, general test cases including sample graphs are present, general class structure is implemented, all team members have knowledge of VANTEDs add-on infrastructure. **Deadline:** Software Design Document 2. Two-Week-Report: Silvan
20.5 - 26.5.	Distribute implementation tasks, first generalized input-output implementations to have first 'hello-world' algorithm/framework, gui skeleton for testing is finished, define basic unit test cases
27.5. - 9.6.	Implementing first simple real algorithms, defining test cases and benchmarks
2.6.	**Deadline:** 3. Two-Week-Report: Thomas
9.6.	**Milestone 2**: First non-comparative implementations of multilevel framework and stress minimization layout algorithm are functional, benchmarks defined
10.6. - 23.6.	Implementation of more complex algorithms with desired asymptotic run-time, testing efficiency
16.6.	**Deadline:** 4. Two-Week-Report: Benjamin
24.6. - 30.6.	Final tests, GUI fine-tuning
30.6.	**Deadline:** 5. Two-Week-Report: Jakob
7.7.	**Milestone 3**: Both plugins are running and were successfully tested
8.7. - 22.7.	Create final presentation and practicing it.
14.7.	**Deadline:** 6. Two-Week-Report: Joshi
23.7.	**Final Presentation**

Table 1: Project Time Schedule