HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Graphs-Visualization-Service GVS 2.0

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Autoren
Michael Wieland
Murièle Trentini

Betreuer
Thomas Letsch
Dozent für Informatik an der
Hochschule für Technik Rapperswil

Zeitraum: 18.09.2017 - 22.12.2017

Abstract

An der HSR (Hochschule für Technik Rapperswil) werden in den Modulen *Algorithmen und Datenstrukturen 1 und 2* diverse Algorithmen unterrichtet, die auf den beiden Datenstrukturen *Graph* und *Tree* arbeiten. Die Funktionsweise solcher Algorithmen ist für Studenten teilweise schwer zu verstehen. Hinzu kommt, dass die Programmierung für die Visualisierung der Algorithmen mit viel Aufwand verbunden ist. Aus diesen Gründen wurde im Jahr 2005, im Rahmen einer Diplomarbeit, der GVS (Graphs-Visualization-Service) 1.0 entwickelt. Der GVS besteht aus einer clientseitigen Software-Bibliothek, sowie einer Server Komponente, welche die eingehenden Datenstrukturen visualisiert. So können Studenten die Arbeitsweise der Algorithmen Schritt für Schritt nachvollziehen.

Diese Studienarbeit beschäftigte sich mit der Erneuerung des GVS 1.0, wobei insbesondere bei der Server Komponente das UI Framework mit JavaFX ersetzt wurde. In der Analysephase wurde die Funktionsweise des Vorgängerprodukts untersucht und Vorschläge für Verbesserungen gemacht. Die gewonnenen Erkenntnisse flossen in eine klare Schichtenarchitektur ein, die es auch Folgearbeiten erleichtern wird, einzelne Schichten zu ersetzen. Ebenfalls wurden zeitgemässe Software Best Practices umgesetzt.

Während dem Projekt konnte eine erweiterbare Software Architektur erstellt werden, die das Vorgängerprodukt vollständig ersetzt. In den eingesetzten Metriken ist klar ersichtlich, dass sich die Code-Qualität über die Dauer des Projekts stark verbessert hat. Zusätzlich profitiert der GVS von einem modernen, intuitiven User Interface, das *Graphen* und *Trees* durch den Einsatz von speziellen Layouting Algorithmen ansprechend darstellt.

Management Summary

Ausgangslage

GVS ist eine Visualisierungs-Software, die Graphen und Trees darstellt. Zusätzlich wird die Arbeitsweise von Algorithmen durch den Einsatz von Farben und gerichteten Kanten visuell hervorgehoben. Die Datenstrukturen werden in der Entwicklungsumgebung des Benutzers implementiert und mit einer clientseitigen Library an die Visualisierungskomponente übertragen. Dort werden die eingehenden Daten interpretiert und auf eine ansprechende Art und Weise dargestellt.

Der im Jahre 2005 entwickelte GVS 1.0 wurde im Unterricht der Module *Algorithmen und Datenstrukturen 1 und 2* aktiv verwendet. Er erleichtert das Erlernen von komplexen Suchalgorithmen und Datenstrukturen, die in den Modulen behandelt werden. Der GVS 1.0 hat aber bestimmte Einschränkungen bezüglich Usability und der Darstellung von grossen Trees. Ebenfalls gilt das eingesetzt UI Framework als veraltet und es existieren in der Zwischenzeit bessere Alternativen.

Vorgehen / Technologien

In einer ersten Phase wurde die bestehende Software auf Funktionsweise und Verbesserungspotential analysiert. Ebenfalls wurde die umgesetzte Architektur kritisch beleuchtet. Da die Software HSR-intern verwendet wird, sind Folgearbeiten nicht auszuschliessen. Es wurde deshalb darauf geachtet, dass mit den gewonnenen Erkenntnissen eine Architektur designet wurde, die einfach verständlich und erweiterbar ist. Zudem sollen zeitgemässe Softwareentwicklungs-Techniken angewendet werden.

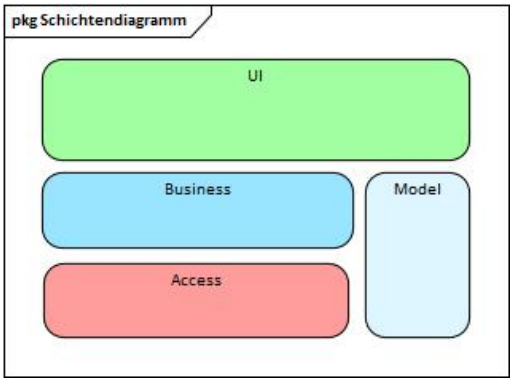


Abbildung 1 – GVS 2.0 Layering

Das in die Jahre gekommene Java Swing GUI Toolkit wurde im Rahmen dieser Studienarbeit mit dem offiziellen Nachfolger JavaFX ersetzt.

Ergebnisse

In der Studienarbeit wurde ein vollständiger Ersatz für den GVS 1.0 entwickelt.
Die Code Qualität konnte durch die Umsetzung einer klaren Schichtenarchitektur und zahlreichen Refactorings gesteigert werden. Dabei wurde ein besonderes Augenmerk auf das Einhalten des Single Responsibility Principle und auf die Reduktion von dupliziertem Code gelegt. Somit konnten sämtliche Metrik-Noten verbessert werden.
Eine Verbesserung der Usability wurde unter anderem durch ein modernes UI mit starken Farbkontrasten und eine automatisch skalierende und zentrierende Zeichenfläche erreicht. Zudem ist GVS 2.0 robuster gegenüber Programmierfehler in der User-Applikation. Bei auftretenden Exceptions springt ein Watchdog ein und kappt die Verbindung zwischen Client und Server.
GVS 2.0 erweitert den bisherigen Funktionsumfang um einen Algorithmus für Trees, welcher fähig ist auch grosse Binary-Trees ansprechend und lesbar zu zeichnen sowie n-ary Trees darzustellen.

Ausblick

Das Projektteam ist vom praktischen Nutzen der Applikation überzeugt und wird sich deshalb in der Bachelorarbeit mit einer generischen Lösung zur Visualisierung von weiteren theoretischen Informatikkonzepten beschäftigen.

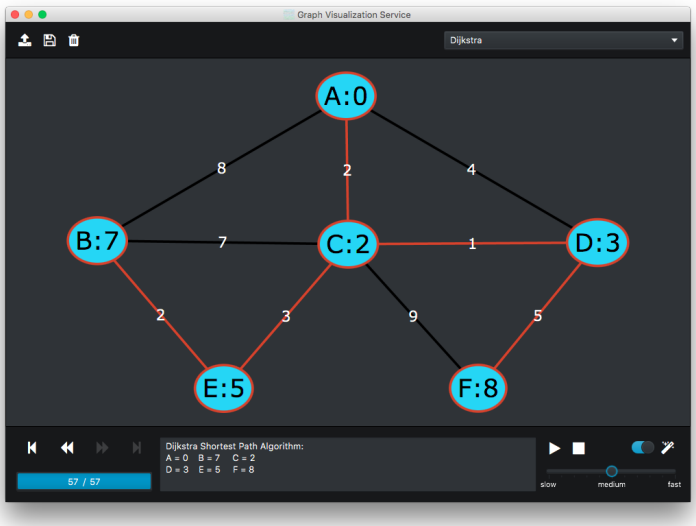


Abbildung 2 – Visualisierter Dijkstra Algorithmus im GVS UI 2.0

Danksagungen

Wir danken folgenden Personen für Ihre Unterstützung:

- Thomas Letsch für die Betreuung und Unterstützung während unserer Studienarbeit
- Prof. Olaf Zimmermann für die interessanten Gespräche während der Architekturphase
- Jessica Martin für die technische Unterstützung beim Logo Design

Inhaltsverzeichnis

Kapitel 1

Ausgangslage

Datum	Version	Änderungen	Autor
28.09.17	1.0	Grundgerüst erstellt	mwieland, mtrentini
18.12.17	1.0	Mehrwert ergänzt	mtrentini

Tabelle 1.1 – Versionshistory Ausgangslage

1.1 Problembeschreibung

An der HSR werden in den Modulen *Algorithmen und Datenstrukturen 1* und *Algorithmen und Datenstrukturen 2* die Funktionsweise von schwierig zu verstehenden Algorithmen unterrichtet. Zusätzlicher Bestandteil der Module ist das Erlernen der beiden Datenstrukturen *Graph* und *Tree*. Die Implementierung derartiger Algorithmen und Datenstrukturen ist oft mit grossem Aufwand verbunden.
Die Software GVS unterstützt Studenten beim Erlernen der beiden Datenstrukturen sowie der darauf anwendbaren Algorithmen. Damit sich die Studenten ausschliesslich auf das Erlernen der Algorithmen fokussieren können, bietet GVS eine Library, welche den Studenten repetitive Implementierungsarbeiten abnimmt. Der wichtigste Aspekt ist jedoch die Visualisierung der Arbeitsweise von Datenstrukturen und Algorithmen. So zeigt die Software zum Beispiel den schrittweisen Aufbau eines Baumes oder den Ablauf des Path Finding Algorithmus "Dijkstra".
Die Software wurde 2005 bereits im Rahmen einer Diplomarbeit umgesetzt. Der in die Jahre gekom-

mene GVS 1.0 soll nun in einem Major-Upgrade aktualisiert werden.

1.2 Mehrwert

GVS wird im Rahmen dieser Studienarbeit auf den neusten Stand der Technik gebracht und profitiert von neueren Java Features wie Generics und Lambdas sowie einer erweiterbaren Schichtenarchitektur.

Für den Endbenutzer ändert sich hauptsächlich das User Interface. Dieses bekommt ein modernes und ansprechendes Design, besitzt intuitiven Zugriff auf vorhandene Features und ist durch die automatisch skalierende und zentrierende Zeichenfläche flexibler in der Darstellung von Graphen und Trees. Zudem hat die Applikation durch das Einführen eines Watchdogs (siehe 4.7) eine höhere Fehlertoleranz.

Durch die Verbesserung der Code Qualität und die Einführung einer sauberen Schichten-Architektur, wird es zukünftigen Entwicklern leichter gemacht, die Applikation zu verstehen und zu erweitern. Zusätzlich wird der Code durch die Nutzung von modernen Lambda-Ausdrücken schlanker und lesbarer.

Kapitel 2

Anforderungsspezifikation

Datum	Version	Änderungen	Autor
17.10.17	1.0	Sequenzdiagramme eingefügt	mtrentini
18.10.17	1.1	Stakeholder eingefügt	mwieland
19.10.17	1.2	Erkenntnisse: Stärken, Schwächen, Schlüsse beschrieben	mwieland

Tabelle 2.1 – Versionshistory Anforderungsspezifikation

2.1 Anforderungen

Die Anforderungen werden massgeblich aus der offiziellen Aufgabenstellung entnommen. (siehe Anhang D)

2.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen werden vollständig von GVS 1.0 [29] übernommen. Der Funktionsumfang der Applikation soll nicht verändert werden. Sollten dennoch Anpassungen am Funktionsumfang nötig sein, werden diese mit dem Betreuer besprochen und in den Sitzungsprotokollen dokumentiert.

2.1.2 Nicht funktionale Anforderungen

1. In der GVS UI Komponente muss das Framework Swing durch JavaFX ersetzt werden.

2. In der GVS Lib Komponente muss die Java und .NET Library um Generics erweitert werden.

3. Punktuell müssen Verbesserungen vorgenommen werden.

Eine genaue Definition der dritten Anforderung ist im Abschnitt Refactoring Ablauf (3.3.1) beschrieben.

2.2 Domainanalyse

In der Domainanalyse wird der GVS 1.0 untersucht. Die Erkenntnisse fliessen in die Designspezifikation ein.

2.2.1 Stakeholder

Stakeholder repräsentieren Personen, die ein Interesse an der vorliegenden Software haben. Aus den funktionalen Anforderungen (2.1) von GVS 1.0 können drei relevant Stakeholder ermittelt werden. Während der Umsetzung wird überprüft, ob die Erwartungen der Stakeholder erfüllt werden können. Wegen der technischen Orientierung der vorliegenden Studienarbeit werden speziell die Ansprüche zukünftiger Entwickler berücksichtigt.

#	Rolle	Ziel/Intention	Erwartungshaltung
1	Student	Möchte GVS unkompliziert installieren und benutzen können. (Unter Zuhilfenahme der Installationsanleitung)	Möchte den zu visualisierenden Graphen/Tree ansprechend dargestellt bekommen. Ihm ist der Lerneffekt wichtig.
2	Entwickler	Möchte GVS einfach erweitern können.	Sucht nach bekannten Software Pattern und klaren Software Schichten. Erwartet eine informative Architekturdokumentation.
3	Dozent	Möchte GVS in seinem Unterricht einsetzen.	Erwartet eine intuitive Integration von GVS in bestehende Übungsaufgaben.

Tabelle 2.2 – Relevante Stakeholder

2.2.2 Aktoren

Aktoren sind externe Einflussfaktoren die mit dem System interagieren. Im GVS 1.0 gibt es zwei Aktoren.

GVS User

Der GVS User interagiert mit dem GVS UI über das User Interface. Er kann die aktuelle Visualisierungs-Session per Knopfdruck auf das Filesystem speichern oder eine bestehende Session von dort laden. Wenn ein Graph visualisiert ist, kann er die Koordinaten der Vertices neu berechnen lassen.

GVS Lib

Die GVS Lib Komponente wird in das Software Projekt des GVS Users eingebunden. Sie sendet in XML (Extensible Markup Language) beschriebene Datenstrukturen über eine Socket Verbindung an das GVS UI. Die Daten werden vom GVS UI auf dem Access Layer empfangen und auf dem User Interface visualisiert.

2.2.3 Systemübersicht

Die Systemübersicht zeigt schematisch wie die beiden Aktoren (Siehe 2.2.2) mit dem die GVS UI interagieren.

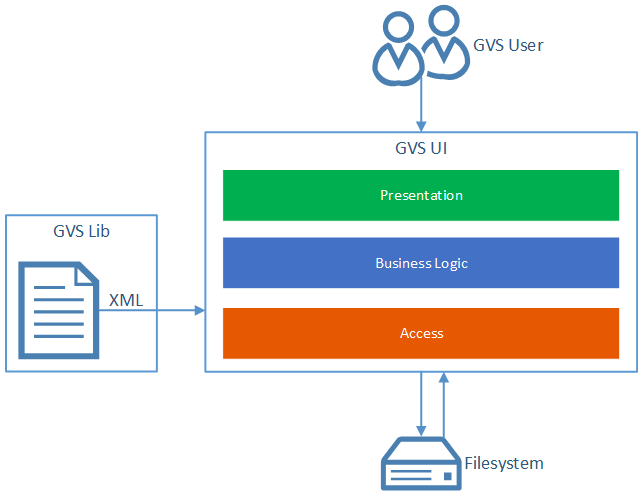


Abbildung 2.1 – GVS: Systemübersicht

2.2.4 Klassendiagramm

Das Klassendiagramm 1 (*Klassendiagramm UI Analyse* im Anhang B) zeigt die wichtigsten Klassen von GVS 1.0. Im vorhergehenden Projekt wurden die Schichten zwar angedacht, sie widerspiegeln sich aber kaum im Package Design. Um den Überblick zu verbessern, sind die Klassen in diesem Diagramm den entsprechenden Schichten zugewiesen. Dies erhöht die Verständlichkeit der Problemdomäne und vereinfacht den Übergang in die neue Schichtenarchitektur von GVS 2.0. Eine genaue Analyse der GVS 1.0 Klassen ist dem Abschnitt 2.2.6 zu entnehmen.

2.2.5 Sequenzdiagramme

Um schwierig zu verstehende Systemabläufe zu abstrahieren, werden Systemdiagramme erstellt. Sie zeigen die wichtigsten Abläufe in GVS 1.0.

Verbindungsaufbau

Das Sequenzdiagramm 2.2 zeigt den Verbindungsaufbau über den *SocketServer* beim Starten der Applikation. Der Start einer *ServerConnectionXML* führt zu einem Locking mit Hilfe des *ConnectionMonitors*. Dieses Locking verhindert, dass mehr als eine GVS Lib gleichzeitig mit einem GVS UI kommuniziert. Nach dem Empfangen der Daten aus dem Socket, erstellt der *ModelBuilder* einen Graphen oder Tree als Domain-Objekt. Dieser Ablauf ist auf dem Sequenzdiagramm **ModelBuilder** (siehe 2.2.5) ersichtlich.

ModelBuilder

Wie im Sequenzdiagramm 2.3 parsed der *ModelBuilder* aus den empfangenen XML Dokumenten das entsprechende Domain-Objekt. Dabei wird zwischen Graphen und Trees unterschieden. Nach Fertigstellung des Parse-Vorgangs, wird das Model an den entsprechenden *SessionController* übergeben. Der weitere Ablauf befindet sich im separaten Sequenzdiagramm **GraphSessionController** (siehe 2.2.5)

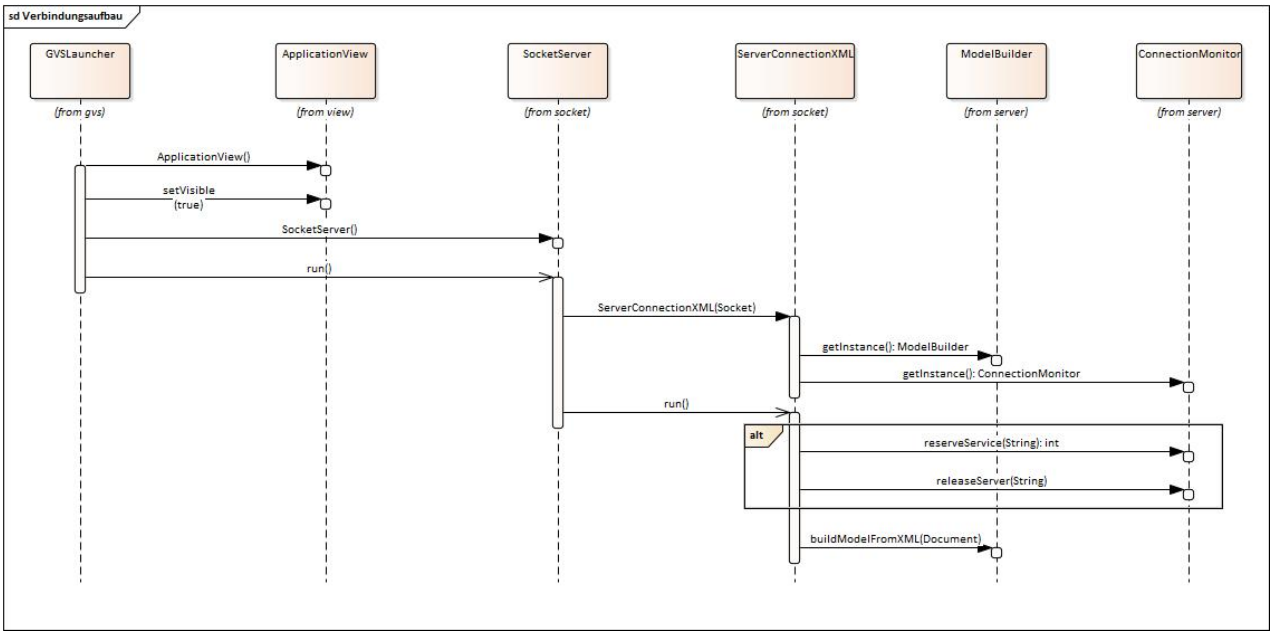


Abbildung 2.2 – Sequenzdiagramm: Verbindungsaufbau

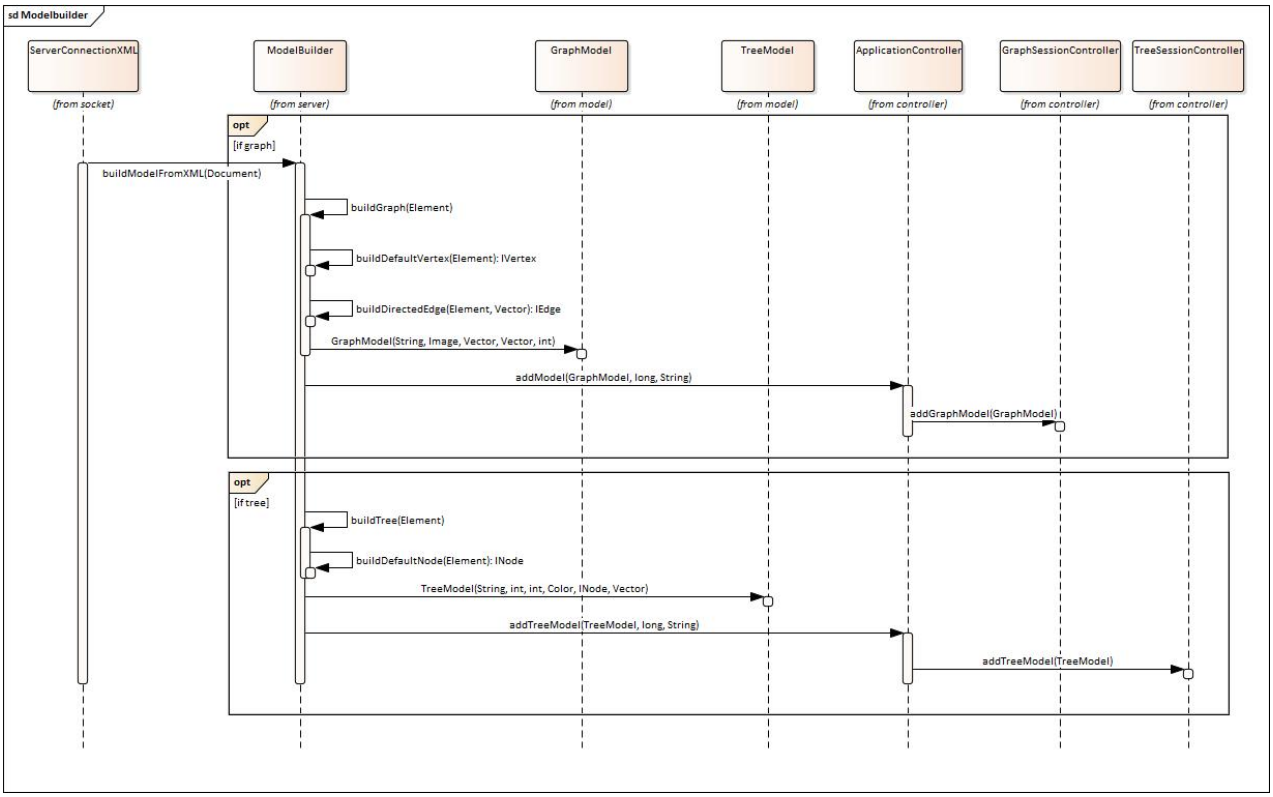


Abbildung 2.3 – Sequenzdiagramm: ModelBuilder

ApplicationView: Load & Save Session

Der Programmablauf beim Laden und Speichern einer Session ist im Sequenzdiagramm 2.4 dargestellt.

Wenn ein User eine Session lädt, wird das entsprechende XML vom *Persistor* geparsed. Je nach Bedarf wird ein Vector aus Graphen oder Trees erstellt. Dieser Vector wird an den *SessionController* übergeben. Der Ablauf innerhalb des *SessionController*s ist im separaten Sequenzdiagramm **GraphSessionController** (siehe 2.2.5) dargestellt.

Nachdem die Session erstellt wurde, wird die *ApplicationView* per Observer-Beziehung (siehe Glossar: Observer Pattern) notifiziert. Sie stellt darauf die neue Session im GVS UI dar.

Auffällig an der Klasse *ApplicationView* ist, dass sie im Constructor einen *Persistor* instanziert. Diese Instanz wird jedoch nicht direkt in der ApplicationView verwendet, sondern wird als Argument an den *ApplicationController* geschickt, welcher dann mit dem Persistor Sessions speichert und lädt. In der Analyse des Projektteams ist unklar geblieben, wieso die Instanzierung des *Persistor*s nicht direkt im *ApplicationController* passiert.

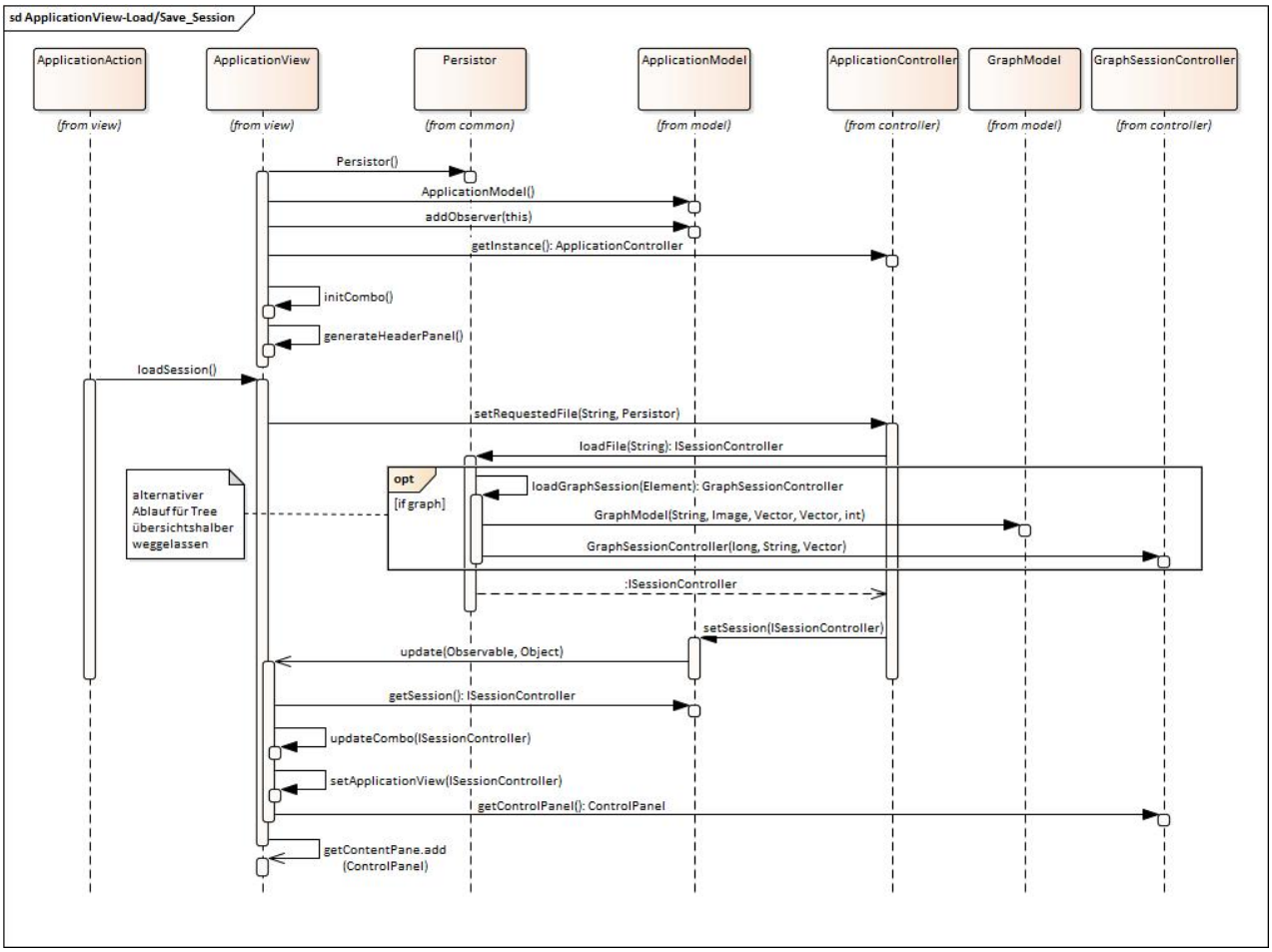


Abbildung 2.4 – Sequenzdiagramm: ApplicationView

GraphSessionController

Wie das Sequenzdiagramm 2.5 zeigt, wird der *GraphSessionController* vom *ModelBuilder* und vom *Persistor* angestossen, wenn ein oder mehrere neue Graphen geparsed wurden. Der *GraphSessionController* ist dafür zuständig diese Graphen in eine neue oder bestehende Session einzufügen. Dabei unterscheidet sich der Programmablauf kaum. Der besseren Übersicht halber ist nur der Fall "ein einzelner neuer Graph wurde empfangen" abgebildet.

Für Graphen die keine fix positionierte Vertices haben, wird in einem weiteren Schritt der *LayoutController* aufgerufen. Dieser steuert die Funktionalität der Physics-Engine. Die Physics-Engine berechnet die Koordinaten von Vertices mittels anziehenden und abstossenden Kräften. (siehe Force Directed Drawing Algorithm) Der *GraphSessionController* wird per Observer-Beziehung (siehe Glossar: Observer Pattern) informiert, sobald die Physics-Engine die Berechnung der Positionen abgeschlossen hat.

Der *GraphSessionController* setzt den neuen Graphen als aktuellen Graphen auf dem *VisualizationGraphModel*. Dieses informiert per Observer-Beziehung (siehe Glossar: Observer Pattern) das *VisualiztionGraphPanel*, welches die für die Darstellung benötigten Vertex und Edge Komponenten erstellt.

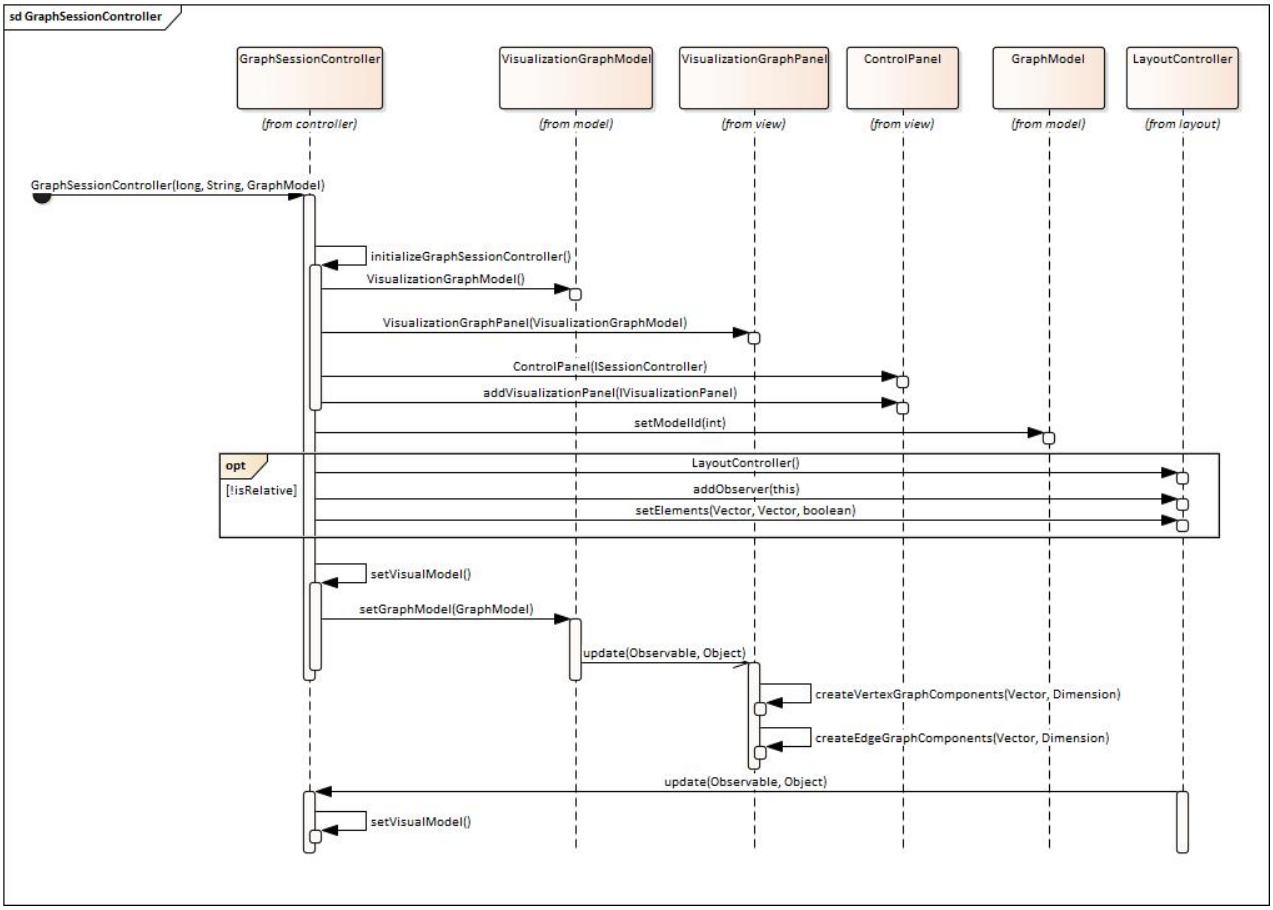


Abbildung 2.5 – Sequenzdiagramm: GraphSessionController

2.2.6 Erkenntnisse

Die aus der Analysephase gewonnenen Erkenntnisse fliessen in die Designspezifikation ein.

Layering

Dem Projektplan GVS 1.0 [29] kann eine Schichtenarchitektur bestehend aus zwei Schichten entnommen werden. Die Schichten widerspiegeln sich aber nicht im Package Design. Gut erkennbar ist das Server Interface, dass die Visualisierung vom Socket Server trennt. GUI spezifische Imports ziehen sich durch mehrere Layer.

Swing/AWT Imports im Business Layer

Im Business Layer gibt es diverse Imports der UI Frameworks Swing rsp. AWT (Abstract Window Toolkit). Dies erschwert einen einfachen Austausch des eingesetzten UI Frameworks.

Tangles

Zwischen dem Presentation und Business Layer existieren Tangles. Dies verhindert einen einfachen Austausch des Presentation Layers.

Magic Numbers

Viele Konstanten sind ungenügend beschrieben. Der Zweck von vielen Zahlen ist daher nur schwer nachvollziehbar.

Namenskonvention

Es gibt keine durchgehende Namenskonvention. Den Variablen fehlt es oft an Informationsgehalt. Ebenfalls sind veränderbare Variablen in Grossbuchstaben geschrieben, was per Konvention [24] den Konstanten vorbehalten ist.

Temporary Fields

Viele Klassen arbeiten exzessiv mit Klassenvariablen, die in diversen Methoden gesetzt und gelesen werden. Des Weiteren werden die Klassenvariablen von Funktionsparameter und lokalen Variablen überdeckt.

Duplicated Code

Klassen wie der *ModelBuilder* und *Persistor* übernehmen sehr ähnliche Aufgaben.

Long Classes

Einige Klassen überschreiten die Obergrenze von 250 Zeilen Code/Klasse. Dies hebt den Verdacht, dass die Klassen mehr als eine Aufgabe erledigen und somit das ”Single Responsibility Principle” verletzen.

Keine Unit Tests

Im GVS 1.0 wurden keine Unit Tests geschrieben. Die Klassen sind sehr schwer testbar, da kein Dependency Injection umgesetzt wurde.

JavaDoc

Das Verhalten von Klassen ist nicht durchgehend dokumentiert. Ebenfalls fehlt es bei einigen Kommentaren an Aussagekraft.

Kapitel 3

Architektur und Designspezifikation

Datum	Version	Änderungen	Autor
13.10.17	1.0	MVVM Konzept geschrieben	mwieland
19.10.17	1.1	Refactoring Konzept erstellt	mtrentini
20.10.17	1.1	Klassendiagramm 2.0	mtrentini, mwieland

Tabelle 3.1 – Versionshistory Architektur und Designspezifikation

3.1 Klassendiagramm

Das Klassendiagramm 2 *"Klassendiagramm Entwurf"* im Anhang B orientiert sich stark an der Vorgängerversion des GVS 1.0. (Siehe 2.2.4). Mit dem Ziel möglichst viel Code wiederzuverwenden, wird der Access und Business Layer mehrheitlich beibehalten. Der Presentation Layer wird vollständig durch die neu eingeführte MVVM (Model View ViewModel) Struktur ersetzt (Siehe 3.2.1).

3.2 Schichtenstruktur

3.2.1 Presentation Layer

Der Presentation Layer wird von der restlichen Anwendung entkoppelt. Damit kann das eingesetzte UI Framework (z.B JavaFX) leicht ersetzt werden. Der Presentation Layer ist nach dem MVVM Prinzip organisiert.

Model-View-ViewModel

MVVM dient der Trennung zwischen dem User Interface und der Anzeigelogik. MVVM ist eine Konkretisierung des verbreiteten MVC (Model View Controller) Pattern und setzt stark auf Databindings. Es wurde von Microsoft für das WPF (Windows Presentation Foundation) Framework entwickelt und wird auch in modernen JavaFX Applikation eingesetzt.

Model Das *Model* enthält reine Datenklassen ohne nennenswerte Logik. (siehe POJO (Plain Old Java Object)) Das *Model* wird in den Business Layer verschoben, damit das UI Framework einfach ausgewechselt werden kann. Werden die Werte im Model geändert, wird das *ViewModel* über eine Observer Beziehung (Observer Pattern) benachrichtigt. Dies entspricht nicht dem klassischen MVVM Konzept, wird jedoch benötigt, damit die View entsprechend aktualisiert wird, wenn z.B die Vertex-Koordinaten neu berechnet werden.

View Die *View* stellt den UI Zustand des *ViewModels* dar. Sie setzt sich aus einer Controller Klasse sowie einer deklarativen FXML (JavaFX XML) Datei zusammen. Die View hält eine Instanz des ViewModels und leitet die eingehenden Anfragen direkt an dieses weiter. Mit der Databinding-API wird eine bidirektionale Kommunikation zwischen *View* und *ViewModel* sichergestellt.

ViewModel Das *ViewModel* kennt die View nicht. Es hat somit keine Abhängigkeiten zu konkreten Anzeige-Elementen. Dies hat den grossen Vorteil, dass die gesamte UI Logik im *ViewModel* gekapselt ist und somit sehr gut testbar ist. Das ViewModel enthält JavaFX spezifische Properties, die für das bidirektionale Binding zwischen UI Komponenten und Code Behind nötig sind [13].

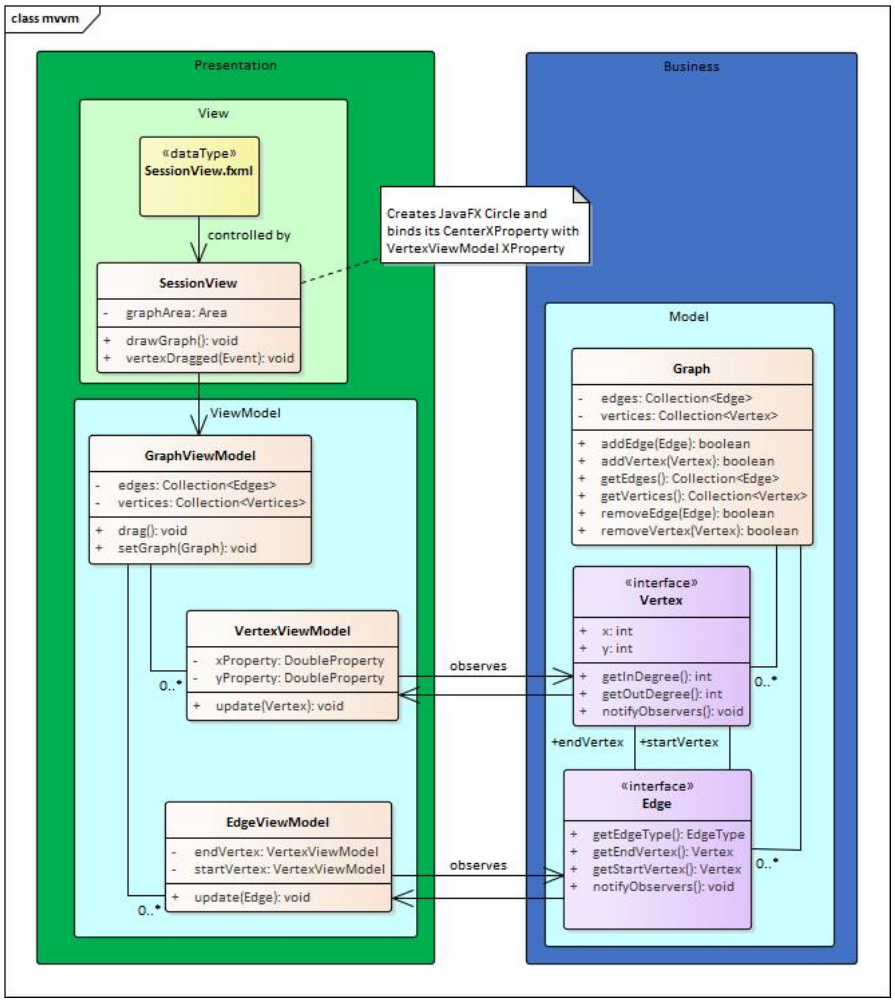


Abbildung 3.1 – Übersicht Model View ViewModel

3.2.2 Business Layer

Im Business Layer liegt die eigentliche Logik. Er enthält keine JavaFX Komponenten, weshalb das eingesetzte UI Framework ohne weiteres ausgewechselt werden kann.

3.2.3 Access Layer

Der Access Layer enthält Klassen für den Zugriff auf den Socket Server, sowie das File System. Es Ist für das Parsen von eingehenden XML Files zuständig und für deren Umwandlung in Domainobjekte.

3.3 Refactoring Konzept

Das Refactoring des GVS 1.0 soll organisiert durchgeführt werden. Dazu werden die notwendigen Schritte priorisiert aufgelistet.

3.3.1 Ablauf

1. Der Presentation Layer wird ersetzt. Dabei wird Swing durch JavaFX ersetzt. Mehr Details finden sich in der Migrations Liste (C) sowie im Klassendiagramm von GVS 2.0 (3.1)
2. Die GVS 1.0 Library wird um Generics erweitert. Dies betrifft die Java-, sowie die .NET Library.
3. Eine klare Schichtenarchitektur (3.2) wird eingeführt. Besonders wichtig ist dabei, das Entfernen von Tangles.
4. Duplicated Code und weitere Code Smells sollen entfernt werden. Ein besonderes Augenmerk wird auf ein konsequentes Naming, das DRY (Don't Repeat Yourself) Konzept sowie das Single Responsibility Principle für Klassen und Methoden gelegt.

3.3.2 Migrations Liste

Die Migrations Liste im Anhang (siehe C) zeigt, welche Klassen aus GVS 1.0 übernommen werden und welche Klassen in welcher Art und Weise verändert oder neu erstellt werden.

Kapitel 4

Umsetzung

Datum	Version	Änderungen	Autor
28.09.17	1.0	Dokument erstellt, Logo beschrieben	mwieland
06.12.17	1.1	Gerichtete Kanten beschrieben	mtrentini
07.12.17	1.2	TreeLayouter beschrieben	mtrentini
08.12.17	1.3	GraphLayouter beschrieben	mwieland
08.12.17	1.4	Multithreading beschrieben	mwieland
08.12.17	1.5	Klassendiagramm GVS 2.0	mtrentini
08.12.17	1.6	Presentation Layer beschrieben	mwieland
10.12.17	1.7	Client Library Update beschrieben	mwieland
11.12.17	1.8	Sequenzdiagramme beschrieben	mtrentini
14.12.17	1.9	Graph & Tree Verschmelzung beschrieben	mtrentini
17.12.17	1.10	Watchdog beschrieben	mtrentini

Tabelle 4.1 – Versionshistory Umsetzung

4.1 Umgesetzte Architektur

4.1.1 Klassendiagramm

Das *"Klassendiagramm UI Umsetzung"* im Anhang B zeigt einen Überblick über die wichtigsten Klassen in GVS 2.0 sowie die umgesetzte Schichtenarchitektur (siehe 3.2). Durch den Einsatz des MVVM Pattern ist der Presentation Layer in sich gekapselt. Im Gegensatz zum GVS 1.0 werden

ausserhalb des Presentation Layers keine UI spezifischen Klassen verwendet. Dies ermöglicht einen Austausch des verwendeten UI Frameworks.

Die folgende Aufzählung (4.1.1) zeigt die wichtigsten Änderungen zum Klassendiagramm aus der Entwurfsphase auf (siehe Anhang B). Ansonsten widerspiegelt GVS 2.0 die angedachte Architektur und Klassenhierarchie.

GvsXmlReader & ClientConnection

Diese Klassen entstanden durch ein Refactoring der GVS 1.0 Klasse *ServerConnectionXML*. Die Trennung dieser Klasse führt dazu, dass das Single Responsibility Principle eingehalten wird.

Watchdog

Behebt das Problem, dass beim Absturz eines Clients die Server Komponenten für andere Clients nicht wieder freigegeben wird. Dies erleichtert das Usability für den Benutzer sehr. Mehr dazu in Abschnitt 4.7

SessionType

Trees und Graphen werden völlig unterschiedlich gelayoutet. Dennoch werden die gleichen Klassen für die Darstellung im Presentation Layer verwendet. Durch das Type Object kann trotzdem ein spezifischer Layouting Algorithmus für jede Datenstruktur eingesetzt werden.

TreeVertex & LeafVertex

In der Designphase wurde das Zusammenführen von Graphen und Trees angedacht. In der Umsetzung hat sich aber gezeigt, dass einerseits die Layouter sowie infolgedessen auch die Vertices unterschieden werden müssen. (siehe 4.6) Für den Tree Layouter (4.4) wird neben dem *TreeVertex* auch ein *LeafVertex* benötigt. Blattknoten dienen als Platzhalter die im UI nicht dargestellt werden. Sie vereinfachen die Berechnung des Tree Layouts aber stark.

4.1.2 Layering

Der GVS 2.0 ist in drei horizontale Layer unterteilt.

Die Zugriffe verlaufen durchgehend von höheren Schichten zu tieferen. Sämtliche Tangles werden mit dem Observer Pattern aufgelöst. Einzige Ausnahme bildet das *Model* Package, dass seitlich am

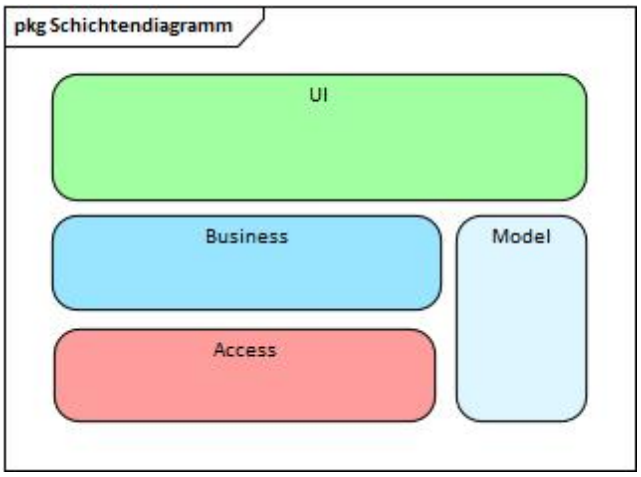


Abbildung 4.1 – GVS 2.0 Layering

Business und *Access* Layer positioniert ist.

Das Strukturanalyse Tool markiert die Zugriffe zwischen *Model* und *Business* leider auch als Tangle (rot). Diese stellen jedoch kein Problem dar, da sich die beiden Packages logisch auf der selben Ebene befinden. Zur besseren Veranschaulichung dient das *"Klassendiagramm UI Umsetzung"* im Anhang B.

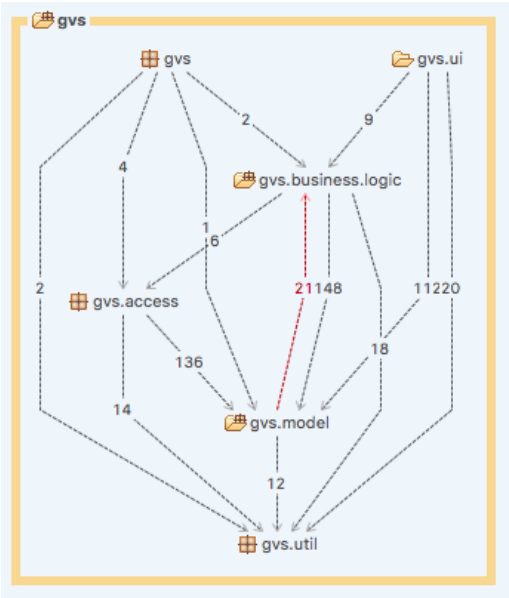


Abbildung 4.2 – GVS 2.0 Package Struktur

4.1.3 Sequenzdiagramme

Während der Analyse-Phase dieses Projekts wurden die Programmabläufe in GVS 1.0 untersucht (siehe 2.2.5). Um Änderungen im Programmablauf aufzuzeigen, finden sich in diesem Kapitel neue Sequenzdiagramme, welche die selben Abläufe in GVS 2.0 abbilden.

Verbindungsaufbau

Abbildung 4.3 zeigt den Start der GVS 2.0 Applikation über die *GVSApplication* sowie das Empfangen von Daten über die Socket. Nicht ersichtlich ist im Diagramm, dass GVS 2.0 an vielen Orten auf Dependency Injection setzt (siehe A.5.1). Zum Beispiel werden die vier Klassen *Watchdog*, *ConnectionMonitor*, *GvsXmlReader* und *ModelBuilder* (rechts im Diagramm) in den Constructor der *ClientConnection* "injected".

Details zu den genauen Änderungen gegenüber GVS 1.0 finden sich im Abschnitt 4.1.1.

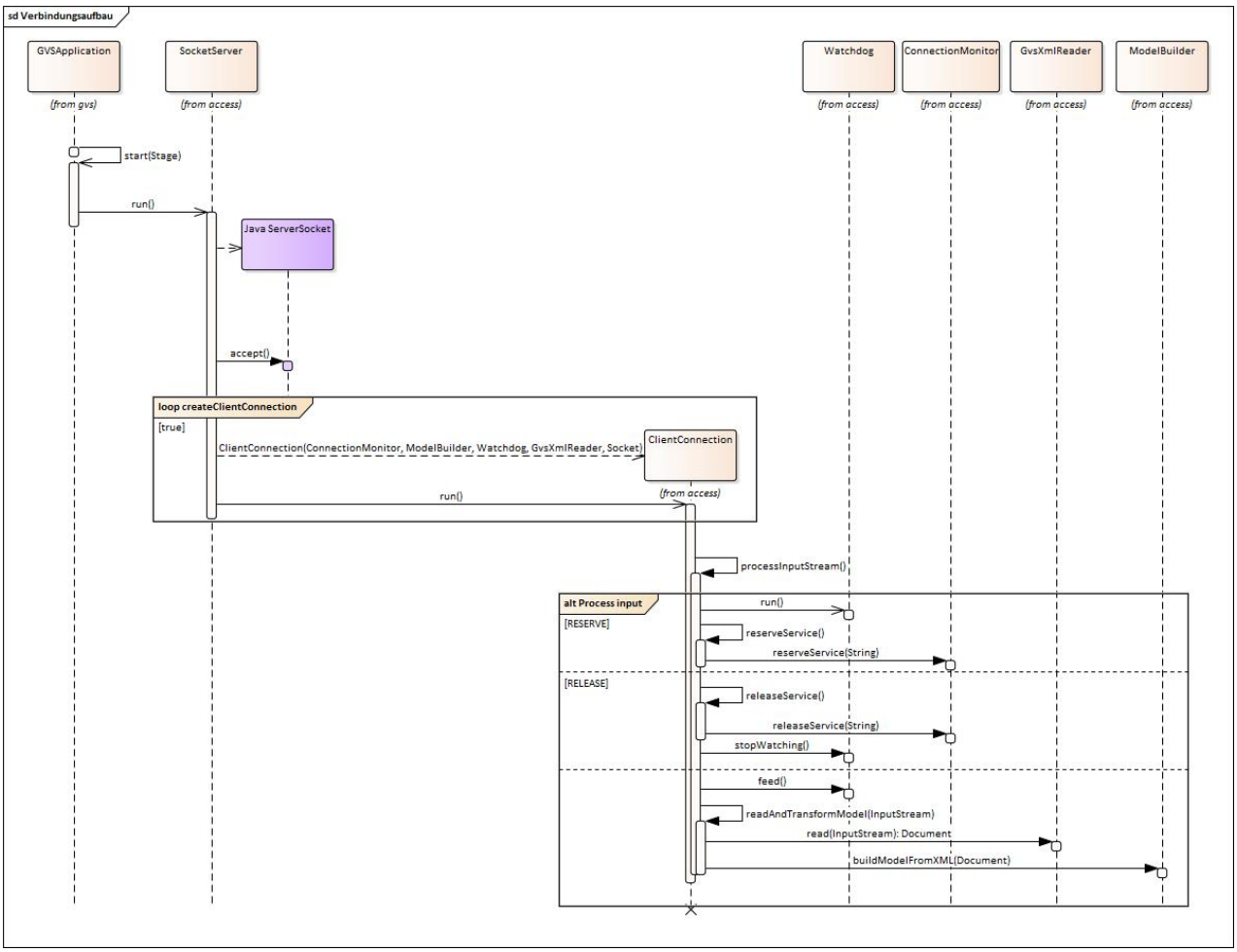


Abbildung 4.3 – Sequenzdiagramm: Verbindungsaufbau

ModelBuilder

Das Sequenzdiagramm in Abbildung 4.4 zeigt, wie aus einem empfangenen XML File POJOs des Business Layers erstellt werden. Sehr gut erkennbar ist, dass im Vergleich zu GVS 1.0 die Trennung zwischen Graph und Tree viel weniger ausgeprägt ist, um *duplicated Code* zu vermeiden (siehe 4.6).

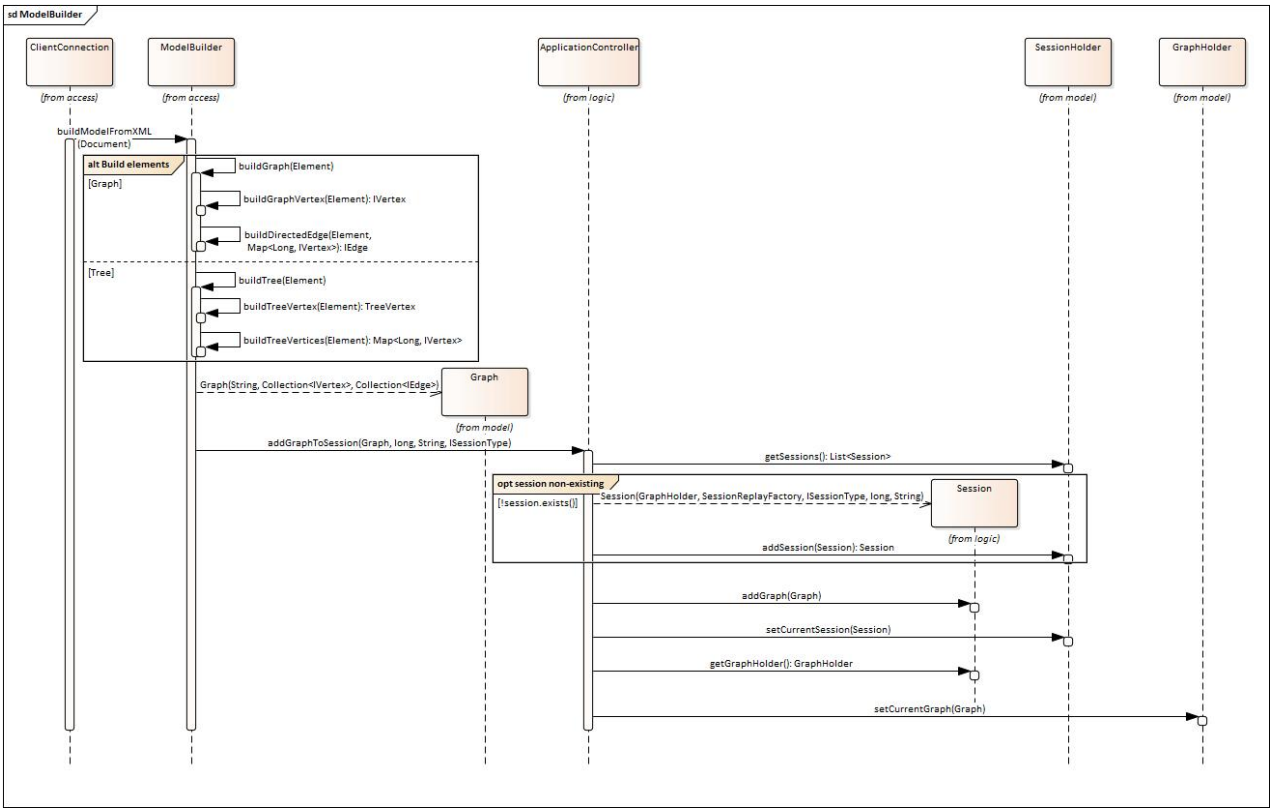


Abbildung 4.4 – Sequenzdiagramm: ModelBuilder

Load & Save Session

Ein User von GVS 2.0 hat die Möglichkeit Sessions zu speichern und vorherige Sessions zu laden. Was sich dabei hinter den Kulissen abspielt, zeigt Abbildung 4.5. Die links eingezeichnete blaue Klasse *JavaFX* widerspiegelt UI Events, welche per *EventHandler* an die *AppView* weitergeleitet werden.

Durch die komplette Ersetzung des Presentation Layers sowie einer klareren Trennung der verschiedenen Schichten unterscheidet sich dieses Sequenzdiagramm stark vom entsprechenden Pendant in GVS 1.0 (2.4). Sie decken jedoch beide die gleichen Use Cases ab.

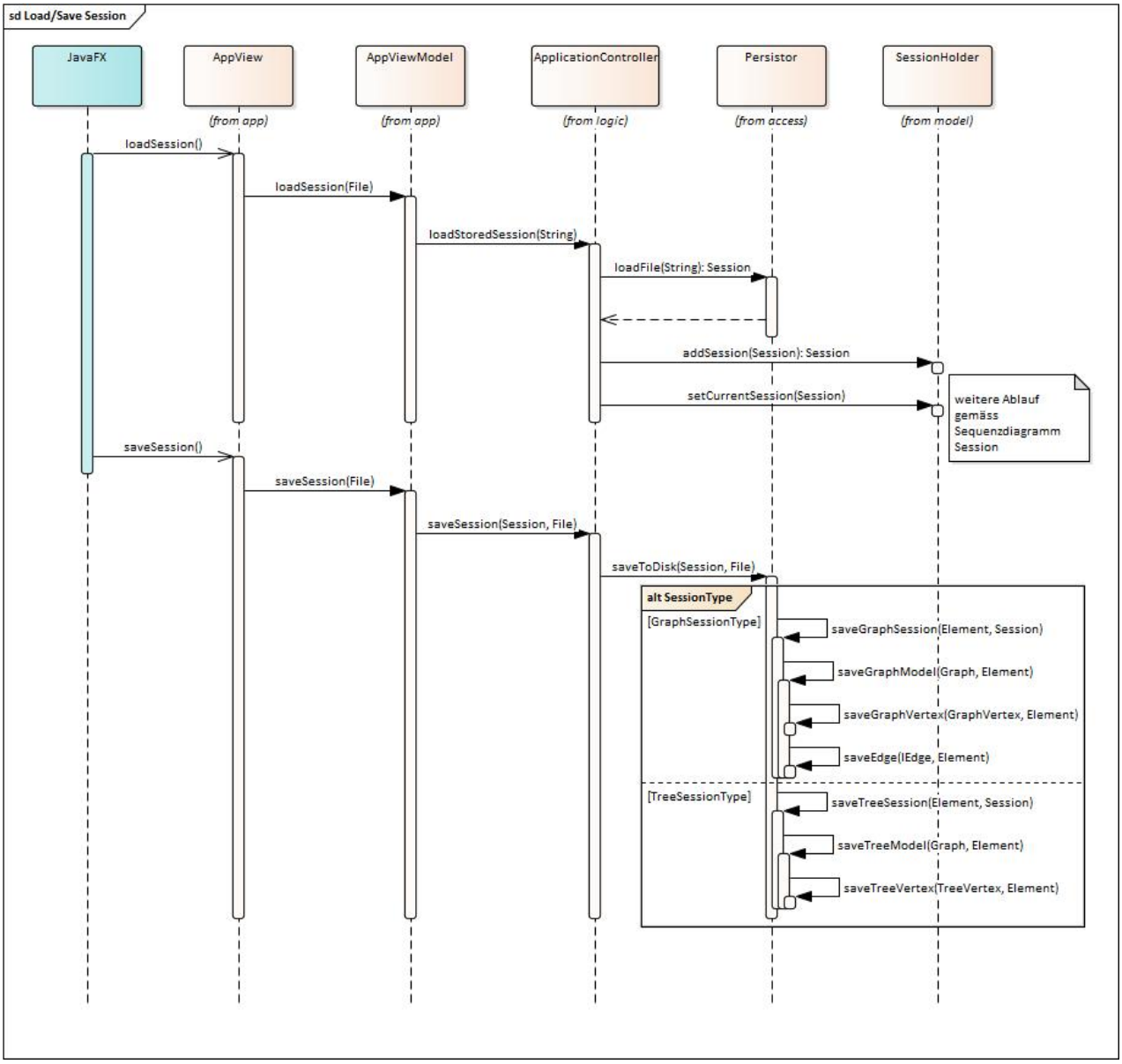


Abbildung 4.5 – Sequenzdiagramm: Load & Save Session

Session

Auch in Abbildung 4.6 zeigen sich die Änderungen im Presentation Layer und die strukturellen Änderungen an der gesamten Applikation stark. So ist nicht auf den ersten Blick ersichtlich, dass Abbildung 2.5 und Abbildung 4.6 die gleiche Programmlogik aufzeigen, nämlich wie POJOs des Business Layers an den Presentation Layer gelangen und auf dem UI dargestellt werden.

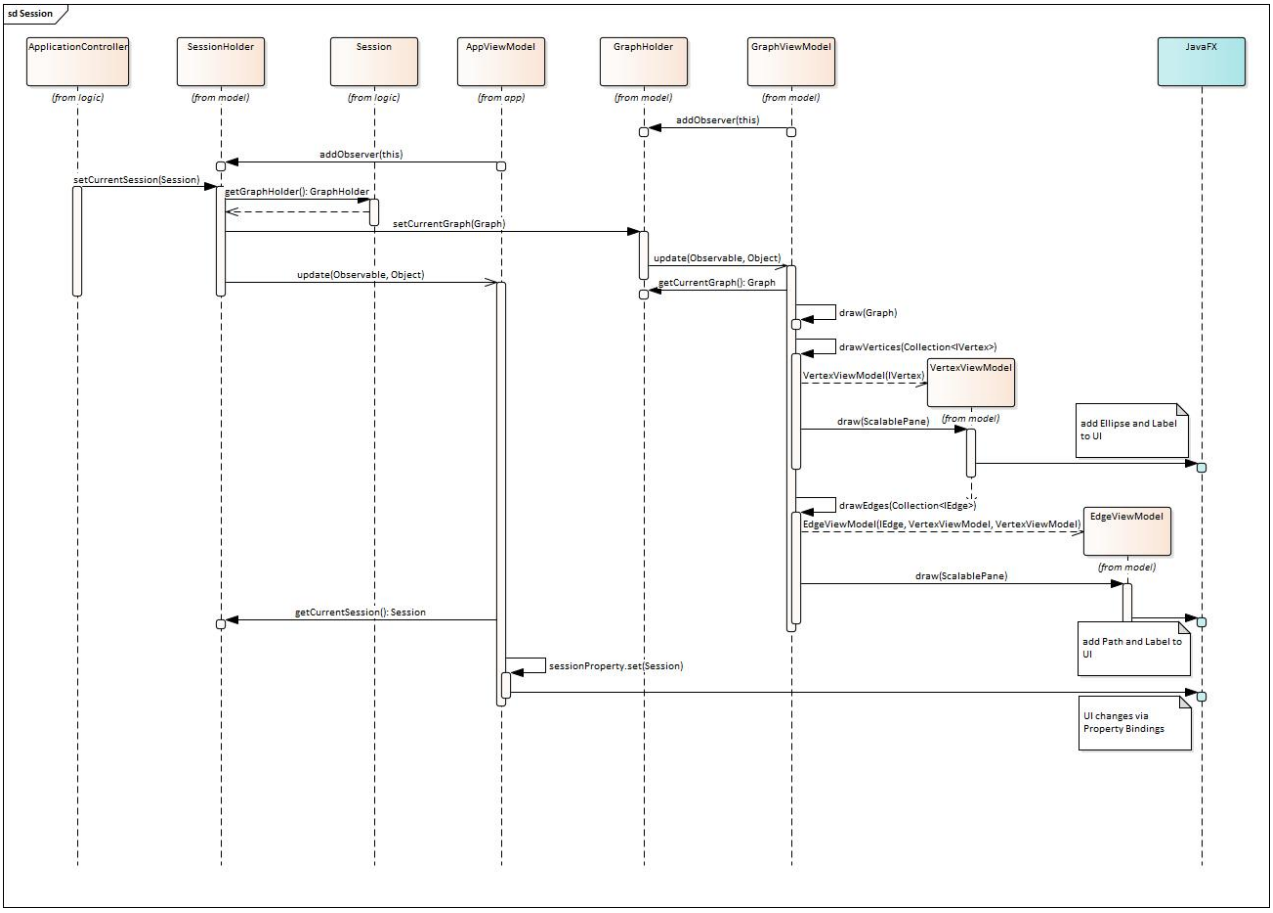


Abbildung 4.6 – Sequenzdiagramm: Session

4.2 Multithreading

Um die Benutzeroberfläche reaktiv zu halten und gleichzeitig auf Datenübertragungen von der Client Lib zu reagieren werden mehrere Threads benötigt. Daher müssen Synchronisationspunkte eingerichtet werden, damit typische Gefahren der Nebenläufigkeit (z.B Race Conditions) nicht auftreten können. Im GVS 2.0 sind folgende Klassen als eigenständige Threads konzipiert.

4.2.1 Nebenläufige Klassen

Tabelle 4.2 zeigt sämtliche nebenläufige Klassen des GVS 2.0.

Klasse	Beschreibung	Lebensdauer
GVSApplication	Läuft im Java Main Thread. Startet den JavaFX UI Thread sowie den SocketServer Thread	Programmstart bis App Terminierung.
UI Klassen	Alle UI Klassen laufen im JavaFX Thread	Programmstart bis App Terminierung
SocketServer	Startet einen *SocketServer* auf einen konfigurierbaren Port und erstellt *ClientConnection* Instanzen für jede eingehende Anfrage.	Programmstart bis App Terminierung.
ClientConnection	Interpretiert mit Hilfe des *GvsXmlReaders* die eingehende Verbindung und leitet die XML Daten an den ModelBuilder weiter	Während einer Client Verbindung
Watchdog	Ermöglicht, dass ein Client den Server nur so lange reserviert wie auch Daten übertragen werden	Während einer Client Verbindung
AreaTicker	Representiert einen Layouting Pulse. Definiert wie oft die View während dem Layouting eines Graphen aktualisiert wird.	Während dem Layout Prozess eines Graphen
SessionReplay	Gibt Graph Snapshots in einem konfigurierbaren Intervall wieder	Solange die Session existiert

Tabelle 4.2 – Übersicht nebenläufige Klassen

4.2.2 Synchronisationspunkte

Die nebenläufigen Klassen werden von speziellen Synchronisationspunkten serialisiert, die in Tabelle 4.3 aufgelistet sind. Die Synchronisationspunkte werden von Guice [20] als Singleton instantiiert, damit alle Threads das selbe Monitor Objekt beziehen. Die Singletons in Guice dürfen keinesfalls mit dem klassischen GoF (Gang of Four) Singleton verglichen werden. Die Guice Singletons besitzen keinen privaten Konstruktor und auch keine statische Instanz der eigenen Klasse. Vielmehr stellt Guice sicher, dass pro Injector genau eine Instanz der Klasse erzeugt wird. Diese Instanz wird dann über den Konstruktor bei allen abhängigen Klassen injiziert. Somit bleibt eine abhängige Klasse testbar, da die Instanz einfach durch einen Mock ersetzt werden kann.
Im Gegensatz zum klassischen Singleton garantiert also der Erzeuger des Sigletons (i.e. Guice Injector), die Einmaligkeit der Instanz, statt die Instanz selber. Dieses Verhalten wird auch *Weak Singleton* genannt, wobei die Begrifflichkeit keine allgemeine Gültigkeit hat.

Klasse	Beschreibung	Lebendauer
ConnectionMonitor	Garantiert, dass immer nur ein Client den GVS Service reservieren kann	Programmstart bis App Terminierung.
ApplicationController	Zentraler Eintrittspunkt für Manipulationen durch das UI sowie durch die Socket Verbindung	Programmstart bis App Terminierung.
GraphLayouter	Überwacht die Lebensdauer des TickerThreads	Programmstart bis App Terminierung.

Tabelle 4.3 – Übersicht Synchronisationspunkte

4.2.3 UI Thread

Wichtig ist, dass alle Änderungen am User Interface ausschliesslich vom JavaFX UI Thread durchgeführt werden. Dazu wird bei allen Eintrittspunkten in den Presentation Layer ein `runLater()` ausgeführt (siehe Code Abschnitt 4.1). Dies veranlasst einen Thread Wechsel zum UI Thread.

Listing 4.1 – Java FX UI Thread

```
@Override
public void update(Observable o, Object arg) {
    // Hand updates over to JavaFX Thread
    Platform.runLater(() -> {
        doUIManipulation();
    });
}
```

4.3 Gerichtete Kanten

4.3.1 Problem

In einem ersten Stadium hat GVS 2.0 nur ungerichtete Kanten angezeigt. Somit konnten Edges sehr einfach als Linien dargestellt werden, die zwei Vertex Mittelpunkte verbinden. Im endgültigen Stadium zeigt das GVS 2.0 auch gerichtete Kanten an. Somit genügt der einfach Lösungsansatz aus Stadium Eins nicht mehr, denn bei diesem Ansatz werden Pfeilspitzen durch die darüber liegenden Vertices verdeckt.

Um das Problem zu lösen, braucht es ein Algorithmus, der die Schnittpunkte zwischen einer Kante und dem Rand eines Vertex findet. Damit kann ein Pfeil zwischen zwei Schnittpunkten statt zwei Vertex Mittelpunkten gezeichnet werden. Zu beachten ist zudem, dass sich diese Schnittpunkte ändern, wenn die betreffenden Vertices vom User ”gedragged” werden.

4.3.2 Lösung

Die Lösung bietet ein Suchalgorithmus, der wie der *Binary Search Algorithm* [19] funktioniert. So wird ebenfalls mit zwei Endpunkten begonnen, welche in diesem Fall den Mittelpunkten des Start- und Endvertex der Kante entsprechen. Mittels Rekursion wird dann der Schnittpunkt gesucht. Beispielhaft ist der Ablauf mit den verschiedenen Iterationen in Abbildung 4.7 dargestellt.
In einem ersten Schritt wird der Mittelpunkt zwischen den beiden Endpunkten bestimmt (in Abbildung 4.7 grün respektive orange eingezeichnet). Dann wird getestet, ob sich der Mittelpunkt innerhalb

oder ausserhalb des Vertex befindet, denn für jede Iteration wird ein Punkt innerhalb und ein Punkt ausserhalb des Vertex benötigt.

Für die nächste Iteration wird der neu bestimmte Mittelpunkt plus einer der Endpunkte übergeben. In Abbildung 4.7 zeigen Iterationen eins bis drei den Fall, dass der Mittelpunkt ausserhalb des Vertex zu liegen kommt. Ab Iteration Vier wechselt das Szenario, da der Mittelpunkt innerhalb des Vertex zu liegen kommt. In Iteration Sechs ist der Abstand zwischen den Endpunkten so klein, dass die Abbruch Bedingung der Rekursion zu tragen kommt und der gesuchte Schnittpunkt gefunden ist.

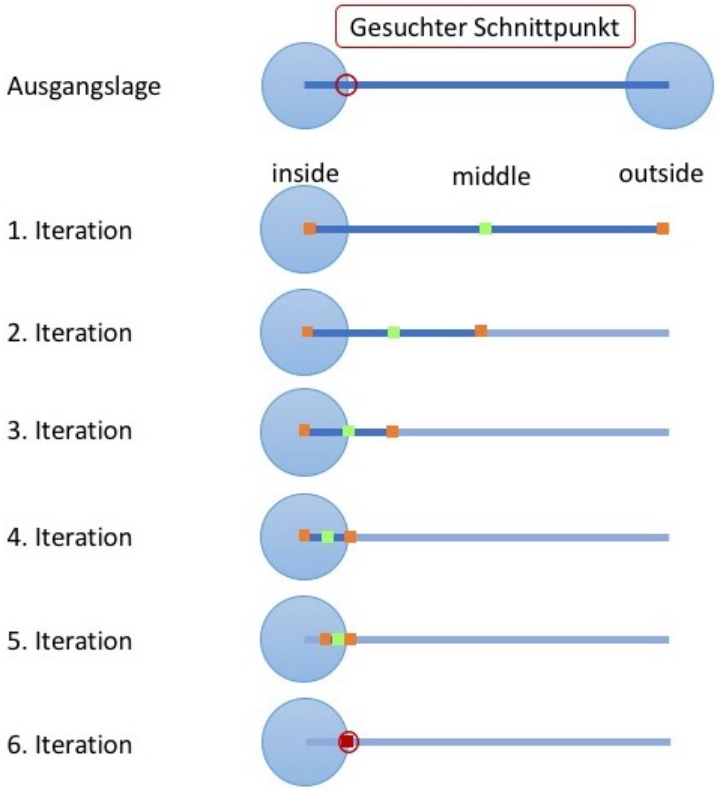


Abbildung 4.7 – Algorithmus zur Suche eines Schnittpunktes

Die spezifische Implementierung dieses Suchalgorithmus ist dem Listing 4.2 zu entnehmen.

Listing 4.2 – Algorithmus Schnittpunkt Suche

```
public Point2D findIntersectionPoint(Point2D outside, Point2D inside) {
    Point2D middle = outside.midpoint(inside);

    double deltaX = outside.getX() - inside.getX();
    double deltaY = outside.getY() - inside.getY();

    if (Math.hypot(deltaX, deltaY) < 1.) {
        return middle;
    } else {
        if (ellipse.contains(middle)) {
            return findIntersectionPoint(outside, middle);
        } else {
            return findIntersectionPoint(middle, inside);
        }
    }
}
```

4.4 Tree Layouter

Die Darstellung von Trees auf eine optisch ansprechende und leserliche Art und Weise bietet verschiedene Schwierigkeiten. Bereits 1981 haben Reingold-Tilford [30] zu diesem Zweck einen Algorithmus entwickelt, welcher binäre Bäume zeichnet. Dieser Algorithmus wurde von Walker weiterentwickelt, um n-ary Trees darzustellen und schlussendlich von Buchheim, Jünger und Leipert [2] verbessert, um die Laufzeit auf $\mathcal{O}(n)$ zu beschränken.

Der von GVS 2.0 verwendete Algorithmus, basiert auf diesem neusten Tree-Algorithmus und genügt somit auch den fünf anerkannten Prinzipien, denen ein solcher Algorithmus genügen muss:

- Das Tree Layout muss die hierarchische Struktur des Trees widerspiegeln. Das heisst konkret, dass die y-Koordinate eines Knoten durch seinen Level gegeben ist.

- Die Knoten eines Levels sind so anzuordnen, dass sich ihre Edges nicht überkreuzen und die Knoten einen gewissen horizontalen Mindestabstand einhalten.
- Das Zeichnen eines Subtrees hängt nicht davon ab, an welcher Stelle im Baum sich der Subtree befindet. Das bedeutet, dass isomorphe Subtrees identisch gezeichnet werden.
- Die Reihenfolge von Kindern muss im Baum ersichtlich sein. Im Falle eines binären Trees heisst das, dass ein linkes Kind links des Eltern-Knoten gezeichnet wird.
- Der Algorithmus arbeitet symmetrisch.

Ein weiterer Vorteil dieses Algorithmus ist, dass er Bäume immer so zeichnet, dass sie eine minimale Breite besitzen. Dadurch werden auch Bäume mit sehr vielen Nodes schön dargestellt. Dies führt dazu, dass der in GVS 1.0 [29] verwendete *Cluster Splitter* nicht mehr benötigt wird.

4.5 Graph Layouter

4.5.1 Physics Engine

Für das Layouting der Graphen wurde ein Force-Directed Drawing Algorithm verwendet. Dieser wurde mehrheitlich von GVS 1.0 übernommen. Der Algorithmus ordnet die Vertices so an, dass es möglichst wenig überschneidende Kanten gibt. Dazu wird für jeden Vertex ein *Particle* Objekt erstellt, welches während der dynamischen Plazierung auf der *Area* bewegt wird. Zu Beginn des Layout Prozess wird ein Particle pseudo-zufällig platziert.

Damit die Änderungen des Layouters in Echtzeit dargestellt werden, läuft der komplette Layouting Prozess in einem eigenständigen Thread (siehe *AreaTicker* Thread in der Tabelle 4.2). Sobald ein Graph gelayouted werden muss, wird ein *AreaTicker* Thread gestartet, der in einem bestimmten Intervall die Darstellung der neuen Koordinaten im UI veranlasst. Es läuft immer nur genau ein *AreaTicker* Thread. Sobald sich die *Particles* in der *Area* stabilisiert haben, wird der *AreaTicker* Thread terminiert. Das Sequenzdiagramm 4.8 zeigt den grundlegenden Ablauf beim Starten des Graph Layout Prozess.

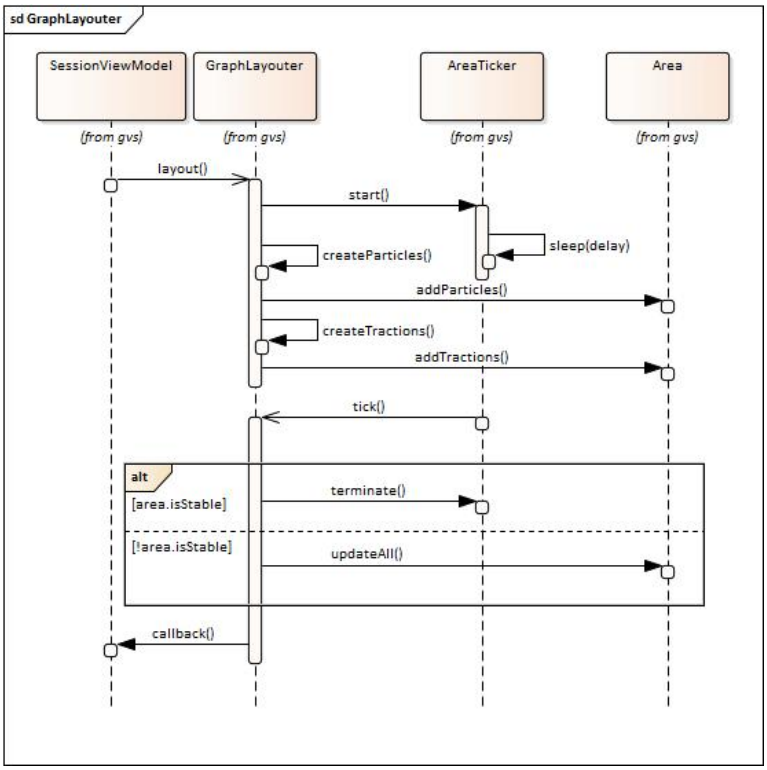


Abbildung 4.8 – Sequenzdiagramm: Graph Layouter

4.5.2 Area

Die *Area* dient als Container während dem dynamischen Layout Prozess. Sie beinhaltet mehrere *Particle*s die von physischen Kräften bewegt werden (*Traction*s und *RepulsiveForce*s). Ein *Particle* repräsentiert einen Vertex. Die Positionen der *Particles* werden fortlaufend berechnet. Sobald ein Tick Event passiert, werden die aktuellen Koordinaten des Particles in den *GraphVertex* im Business Layer kopiert. Über eine Observer Beziehung wird das *VertexViewModel* im Presentation Layer über die neuen Koordinaten notifziert. Dessen X/Y Properties sind über ein bidirektionales Binding mit dem UI Element (z.B *Ellipse*) verbunden. Die neuen Koordinaten werden auf der Oberfläche dargestellt.

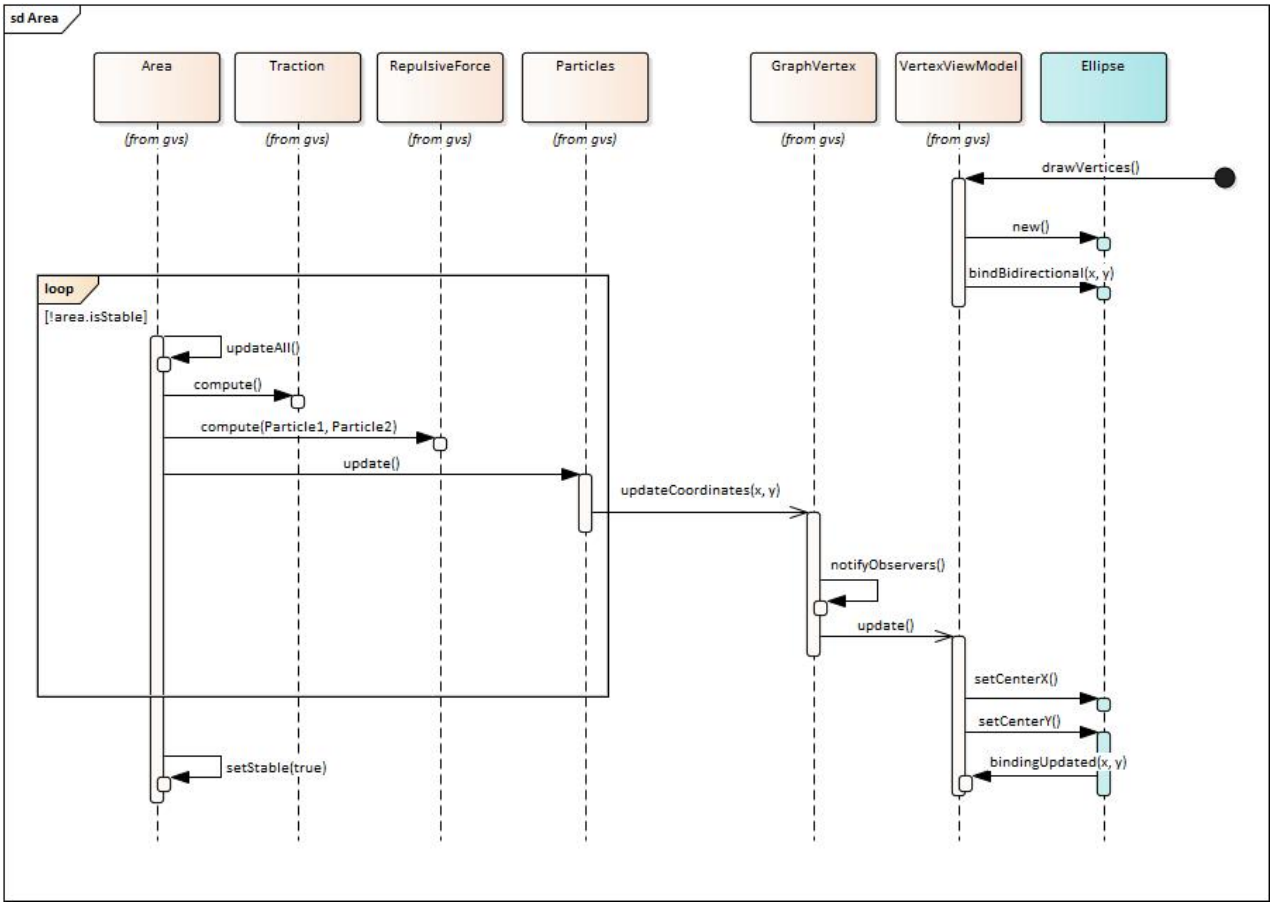


Abbildung 4.9 – Sequenzdiagramm: Area

4.6 Graph und Tree Verschmelzung

In theoretischer Hinsicht sind Bäume Spezialformen von Graphen. In GVS 1.0 wurden diese beiden Strukturen jedoch durchgehend getrennt modelliert. GVS 2.0 vereinigt diese Strukturen soweit es technisch Sinn macht, mit dem Ziel, Code Duplikationen weit möglichst zu vermeiden.

4.6.1 Unterschiede zwischen Graphen und Trees

- Ein Graph besitzt Edges und Vertices. Ein Baum hingegen besitzt nur Vertices, denn seine Edges sind durch die Vater-Kind Beziehung der Vertices implizit gegeben.
- Ein Tree Vertex besitzt zusätzliche Attribute wie Vater-Kind Beziehungen und *isRoot()*
- Die benötigten Layout Algorithmen unterscheiden sich stark. (siehe 4.5.1 für GraphLayouter und 4.4 für TreeLayouter)

4.6.2 Umsetzung

Auf Grund der oben genannten Unterschiede, bildet GVS 2.0 diese Strukturen wie folgt ab:

- Trees und Graphen werden in der Struktur *Graph* vereinigt. Der *ModelBuilder* erstellt für empfangene Bäume sämtliche impliziten Edges.
- Ein *Graph* besitzt eine Liste von Vertices. Diese müssen das Interface *IVertex* implementieren. Es gibt drei konkrete Implementierungen dieses Interfaces. Für reine Graphen wird die Klasse *GraphVertex* benutzt. Die Klassen *TreeVertex* und *LeafVertex* bieten Tree-spzifische Attribute.
- Eine *Session* umfasst jeweils mehrere *Graphen*. Es gibt keine Unterscheidung zwischen Graph-Sessions und Tree-Sessions. Um einer *Session* den richtigen *Layouter* zuzuweisen, besitzt sie einen *SessionType* der beschreibt, ob die enthaltenen Graphen effektive Graphen oder Trees sind.
- Nach dem Abschliessen des Layoutingprozesses ist keine Unterscheidung zwischen Trees und Graphen mehr nötig. Dies führt dazu, dass die Strukturen im Presentation Layer vollständig zusammengeführt sind.

4.7 Watchdog

4.7.1 Bedeutung in der Informatik

Im Allgemeinen versteht man in der Informatik unter einem Watchdog eine System Komponente, die andere Komponenten überwacht. Wenn in der überwachten Komponente ein Fehler auftritt, wird dieser vom Watchdog erkannt und er leitet eine angemessene Fehlerbehebung ein.

Watchdogs werden vor allem in der Hardware und in der Software von Microcontrollern eingesetzt, um einen kompletten Ausfall des vom Microcontroller gesteuerten Geräts vorzubeugen.

4.7.2 Arbeitsweise

Während dem normalen Betrieb einer Software, wird der Watchdog regelmässig vom zu überwachendem System gefüttern. Das bedeutet, der Timer des Watchdogs wird zurückgesetzt. Beim Auftreten eines Fehlers verzögert sich diese "Fütterung" oder fällt gar ganz aus. Dies führt zum Ablaufen des Watchdog Timers.

Sobald der Timer ausgelaufen ist, wird der Watchdog aktiv und leitet Fehlerbehebungsmassnahmen ein.

4.7.3 Watchdog in GVS 2.0

Problem

Da per Spezifikation immer nur ein Client gleichzeitig mit dem Server kommunizieren darf, startet jeder Verbindungsaufbau mit dem Reservieren des GVS UI Services. Um den Service für andere Clients wieder frei zu geben, muss der Client nach dem Übertragen der Daten ein *Release Command* senden.

Der Einsatz des GVS im Unterricht findet relativ früh im Studium statt. Deshalb kommt es immer wieder einmal vor, dass sich Fehler in den Code einschleichen, welcher die GVS Lib benutzt. Das Auftretten einer Programm-beendenden Exception führt dazu, dass das *Release Command* nicht

gesendet wird, was den GVS Service endlos blockiert. Für den Endbenutzer ist nicht ersichtlich, wieso das GVS UI nicht auf den User-Input reagiert. Einziger Ausweg bleibt, die Applikation über den Task-Manager zu beenden.

Lösung

GVS 2.0 führt einen Watchdog ein, der die Verbindung des Clients überwacht. Bei jedem Eintreffen neuer Daten, wird der Watchdog gefüttert bis er beim Erhalten des *Release Command*s beendet wird. Wenn nun das Auftreten einer Exception im Client Code dazu führt, dass der *Release Command* nicht gesendet wird, läuft der Timer des Watchdogs aus. Dadurch wird der Watchdog aktiv und forciert das Freigeben des Services. Dieser Vorgang ist in Abbildung 4.10 visualisiert.

Der Endnutzer nimmt die Komplexität dieses Vorgangs nicht wahr. Nach einer Exception kann das Benutzer Programm einfach erneut gestartet werden und das GVS UI zeigt wie Erwartet die neue Session an.

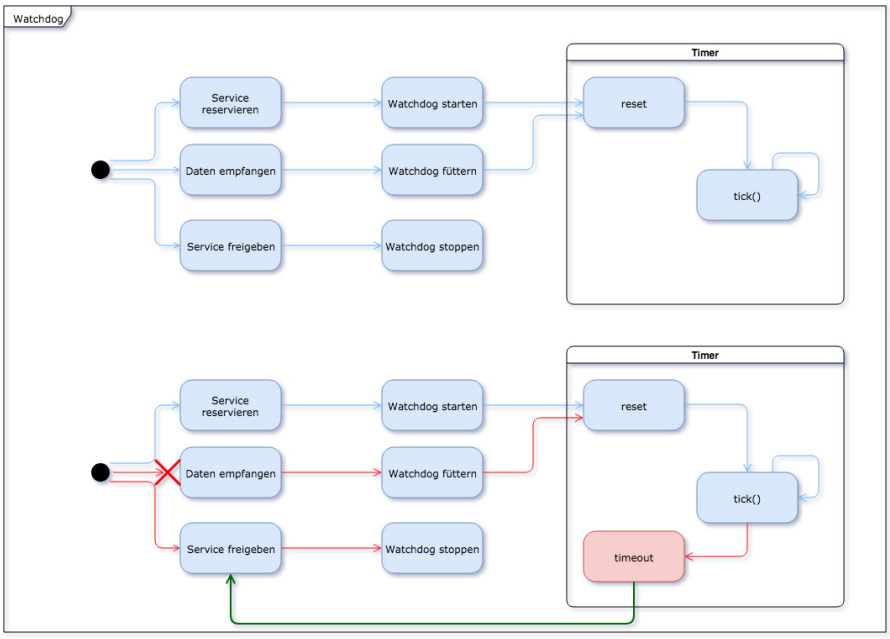


Abbildung 4.10 – GVS 2.0: Watchdog

4.8 Presentation Layer

4.8.1 ScalablePane

Während der Umsetzungsphase hat sich gezeigt, dass die Anzahl an Vertices in Trees und Graphen einen grossen Einfluss auf ihre Darstellung hat. Um diesen Ansprüchen gerecht zu werden, bietet GVS 2.0 eine Zeichenfläche, welche ihre Inhalte automatisch skaliert und zentriert. Dadurch werden zum Beispiel grosse Trees leserlich abgebildet und kleine Graphen verwenden die gesamte Bildschirmbreite.

4.8.2 Drag Support

Bei Graphen können alle Vertices beliebig mit der Maus positioniert werden. Trees können nicht manuell positioniert werden. Eine entsprechende Meldung wird angezeigt.

4.8.3 Autolayout

Das Autolayout steht nur für Graphen zur Verfügung. Die Vertices werden vom Graph Layouter so positioniert, dass es möglichst wenig Überschneidungen der Edges gibt. Standardmässig ist die Einstellung *Force Layout* aktiv. Dies führt dazu, dass beim Betätigen des Auto-Layout Buttons die Koordinaten sämtlicher Vertices neu berechnet werden. Das Deaktivieren dieser Einstellung hat zur Folge, dass Vertices, die manuell durch den User positioniert wurden, nicht von einem Auto-Layout betroffen sind. Sollte dies auf alle Vertices zutreffen, wird der User über einen Tooltip entsprechend informiert.

4.8.4 UI Design

Das User Interface orientiert sich am Aussehen des GVS 1.0. Neu sind alle wichtigen Funktionen direkt über die Toolbar zugreifbar und alle Buttons verfügen über Tooltips. Ebenfalls wurden die Zahlen des Replay Sliders durch sprechende Namen ersetzt.

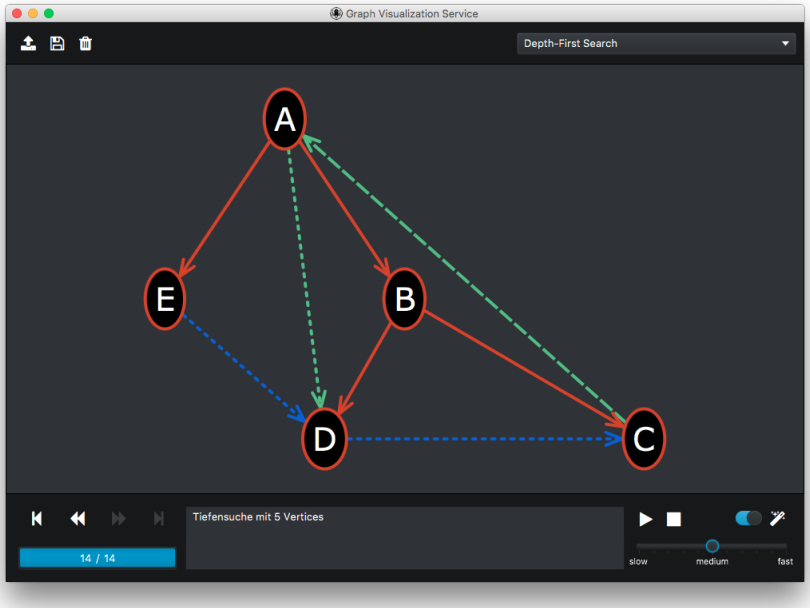


Abbildung 4.11 – GVS 2.0: User Interface

Logo

Das Logo wurde von den Eigenschaften des Kraken [25] inspiriert. Kraken sind bekannt dafür, dass sie Irrgarten-Probleme effizient lösen können. Dies ist eine Anspielung an die Algorithmen, die vom GVS 2.0 unterstützt werden. Ebenfalls wurden die Saugnäpfe des Kraken als Graph Nodes visualisiert und auf der Stirn ist ein binärer Baum zu erkennen.

Abbildung 4.12 – Graphs-Visualization-Service Logo

Farbwahl

Für den Hintergrund kommen bewusst dunkle Farben zum Einsatz. Dadurch werden z.B die gefundenen Pfade eines Algorithmus verstärkt farblich hervorgehoben.

4.9 Client Library Upgrade

Für den GVS stehen Client Libraries für Java und C# zur Verfügung. Beide Clients wurden aufgeräumt und in ein neues, versioniertes Projekt verschoben.

4.9.1 Generics

Gemäss der Aufgabenstellung (siehe Anhang D) müssen beide Clients um Generics erweitert werden. Es hat sich aber gezeigt, dass Generics auf der Schnittstelle keinen Mehrwert bieten würden (Beschluss in Sitzung Woche 11). Schlussendlich wurden deshalb nur die intern verwendeten Listen und Maps um Generics erweitert.

4.9.2 Styles und Icons

Die von GVS 1.0 unterstützten Style-Typen wurden durch einen generellen GVS 2.0 Style ersetzt. Die Typ Klassen aus GVS 1.0 haben nur marginale Unterschiede untereinander. Mit der generellen *GVSStyle* Klasse sowie vier weiteren Enums, konnte damit sehr viele duplizierter Code entfernt werden. Des Weiteren wurden die GVS 1.0 Icons durch FontAwesome Icons ersetzt. Eine Übersicht der migrierten Klassen ist der Tabelle 4.4 zu entnehmen.

4.9.3 C# Lib Refactoring

In der C# Library wurden die gleichen Änderungen wie in der Java Library durchgeführt. Zusätzlich wurde der Code um aktuelle C# Best Practices erweitert. So werden durchgehend implizit typisierte Variablen (`var`) verwendet, sowie der Null-Coalescing Operator (`??`) eingesetzt.

4.9.4 Layering

Auch in den Client Libraries wurde ein einfaches Layering eingeführt. Die Umsetzung der Schichtenarchitektur ist dem Klassendiagramm aus Anhang B zu entnehmen.

Klasse GVS 2.0	Klasse GVS 1.0
GVSStyle	GVSEdgeTyp GVSNodeTyp GVSGraphTyp GVSVertexTyp GVSEllipseVertexTyp GVSEdgeTyp
GVSColor	GVSEllipseVertexTyp.FillColor GVSDefaultTyp.LineColor
GVSIcon	GVSIconVertexTyp
GVSLineStyle	GVSDefaultTyp.LineStyle
GVSLineThickness	GVSDefaultTyp.LineThickness

Tabelle 4.4 – GVS Lib: Übersicht Style Klassen

4.10 Weitere Änderungen gegenüber GVS 1.0

Dieser Abschnitt umfasst kleinere Änderungen, Neuerungen und Fehlerbehebungen gegenüber dem Vorgänger Produkt [29].

4.10.1 Änderungen

MaxLabelLength

In GVS 1.0 wurde eine MaxLabelLength vom Client an den Server übermittelt. In GVS 2.0 ist dieser Wert in der *Configuration* Klasse als Konstante definiert. Da er vor allem Einfluss auf das Layout der Trees hat, ist dieser Wert in GVS 2.0 nicht mehr konfigurierbar. Erhält der GVS 2.0 zu lange Labels, werden diese automatisch durch Auslassungspunkte in der Mitte des Labels gekürzt.

Background

Für Graphen bestand in GSV 1.0 die Möglichkeit, ein Background Bild festzulegen. Da diese Funktionalität im Unterricht nicht verwendet wird, wurde sie nach Rücksprache mit dem Betreuer ersatzlos entfernt.

Icon & IconVertex

Anstelle der Icon Bilder, die für Knoten in GVS 1.0 ausgewählt werden konnten, bietet GVS 2.0 neu sämtliche Icons von FontAwesome [15] an. Zudem gibt es serverseitig keine Unterscheidung zwischen Vertices mit und ohne Icon, was die Klasse *IconVertex* überflüssig macht.

CORBA

Gemäss Vereinbarung mit dem Betreuer wurde die Verbindungsoption über CORBA entfernt.

Log Level Konfiguration

Die Log Level können in GVS 2.0 nicht mehr über das User Interface konfiguriert werden. Stattdessen kann das Log Level mittels JMX (Java Management Extensions) und der JConsole oder durch Anpassen der Logback Konfiguration verändert werden. Dieses Vorgehen ist im Benutzerhandbuch (siehe Anhang G) genauer beschrieben.

4.10.2 Neuerungen

SnapshotDescription

Damit einzelne Schritte in den dargestellten Graphen besser verständlich sind, bietet GVS 2.0 die Möglichkeit kurze Beschreibungen zu erfassen. Diese können serverseitig erfasst, gespeichert und geladen werden. Die Erweiterung des Verbindungsprotokoll, um SnapshotDescription auch clientseitig zu unterstützen, sollte für ein allfälliges Folgeprojekt geplant werden (siehe Kapitel 5.3).

n-ary Trees

Der Tree Layout Algorithmus von GVS 2.0 ist fähig neben Binary Trees auch n-ary Trees zu zeichnen (siehe 4.4). Da der nötige Code clientseitig bereits vorhanden war, konnte das GVS UI mit kleinen Aufwand erweitert werden. Somit können in GVS 2.0 nun auch allgemeinere Trees, wie z.B. Tries, dargestellt werden.

4.10.3 Fehlerbehebungen

Watchog

In GVS 1.0 haben Exceptions im Client Code (der die GVS Lib benutzt) nicht zu einem Verbindungsabbruch geführt. Der Server blieb reserviert und ein neu gestarteter Client konnte keine Verbindung aufbauen. Für den User war nicht ersichtlich, wieso der Server auf den neuen Input nicht reagiert. Um dieses Usability Problem zu beheben bietet GVS 2.0 einen Watchdog Thread. Dieser überwacht die Verbindung und erzwingt einen Verbindungsabbruch, wenn sich der Client längere Zeit nicht mehr meldet. Mehr zum Watchdog unter 4.7.

Speicherort

Der Speicherort für Sessions war in GVS 1.0 hart kodiert. Dies führte auf UNIX Systemen zu einem Fehlverhalten. In GVS 2.0 kann der User den Speicherort für seine Sessions selber wählen.

Entfernung der Root

Die GVS Lib bietet zwei Unterschiedliche Strukturen um Trees darzustellen. Bei Benutzung der Struktur *GVSTreeWithRoot* kam es zu Fehlern, wenn die Root auf *null* gesetzt wurde. In GVS 2.0 ist dieser Fehler behoben.

Kapitel 5

Ergebnisdiskussion

Datum	Version	Änderungen	Autor
11.12.17	1.0	Ausblick geschrieben	mtrentini
11.12.17	1.1	Metriken geschrieben	mwieland
15.12.17	1.2	Zielerreichung geschrieben	mtrentini

Tabelle 5.1 – Versionshistory Ergebnisdiskussion

5.1 Zielerreichung

Dieses Kapitel reflektiert die Erreichung der gemäss Aufgabenstellung (siehe Anhang D) gesetzten Ziele dieser Arbeit. Neben den subjektiven Einschätzungen des Projektteams dienen die benutzten Metriken als handfeste Grundlage für qualitative Verbesserungen.

5.1.1 Ersetzung des Presentation Layers

GVS 1.0 benutzt UI-spezifische Swing und AWT Klassen in sehr vielen Komponenten und durch alle Layer hinweg (siehe 2.2.6). Insbesondere deshalb war das Ersetzen des Presentation Layers keine kleine Aufgabe. Schnell hat sich gezeigt, dass weniger Code als erwartet wiederverwendet werden kann. Im Zuge dieser Ersetzungen von Swing und AWT Klassen durch GVS spezifische Klassen, wurden auch viele kleinere Refactorings umgesetzt. Diese Umstände haben auch dazu geführt, dass

es zu Abweichungen gegenüber der geplanten Migration der Klassen gekommen ist (siehe Anhang C). Die neue Architektur von GVS 2.0 konnte sämtliche Tangles entwirren (siehe 4.1.2) und glänzt mit einem in sich gekoppelten Presentation Layer.

Für den Endbenutzer wirkt das neue User Interface frisch und modern. Durch die direkte Erreichbarkeit aller Funktionen über die Toolbar, ist es auch benutzerfreundlicher als das UI des Vorgängers.

5.1.2 Einführung von Generics in GVS Lib

Im Laufe des Projekts wurde erkannt, dass die GVS Lib bereits einige Generics-Neuerungen enthält und eine weitere Einführung derselben keinen namhaften Mehrwert bringen würde. Somit wurde in Rücksprache mit dem Betreuer entschieden, dass diesbezüglich keine Änderungen an der GVS Lib nötig sind.

Stattdessen konnten die zahlreichen Style-Typ Klassen von GVS 1.0 auf eine einzige Style Klasse (analog zu GVS UI) reduziert werden, was wiederum zukünftige Erweiterungen der Applikation erleichtert.

5.1.3 Punktuelle Verbesserungen

Wie im Kapitel 4 beschrieben, konnten zahlreiche Verbesserungen durchgeführt werden. Von besonderem Nutzen sind hierbei zwei Verbesserungen. Einerseits die Umsetzung einer klaren Schichtenarchitektur, die zukünftigen Entwicklern das Ersetzen einzelner Komponenten erleichtern wird. Andererseits das Einführen eines Watchdogs, welcher die Usability für Studenten stark erhöht, da Exceptions im Implementations-Code nicht mehr zur Blockierung des GVS UI führen.

5.1.4 Bedürfnisse der Stakeholder

Wie in Kapitel 2.2.1 beschrieben, gibt es für den GVS drei relevante Stakeholder. Das in dieser Arbeit entstandene Endprodukt deckt die Bedürfnisse dieser Stakeholder ab.

- Für Studenten entstand eine modern anmutende und intuitiv bedienbare Applikation mit verbesserter Fehlertoleranz (siehe Kapitel 4.7 *Watchdog*).

- Zukünftigen Entwickler wird die Erweiterung oder Ersetzung einzelner Teile der Applikation durch die umgesetzte Schichtenarchitektur erleichtert.
- Der Dozent kann die Applikation wie gewohnt einsetzen. Sämtliche bisherigen Übungen die auf GVS setzen, finden sich in einer erneuerten Version im *Testers* Repository [35].

5.2 Software Metriken

In der Analyse Phase wurden mehrere Code Smells in GVS 1.0 erkannt (Siehe Abschnitt 2.2.6). In der Design Phase wurde deshalb ein Konzept ausgearbeitet, damit das Refactoring strukturiert ablaufen wird (Siehe Abschnitt 3.3). Die Refactorings wurden damals aus Zeitgründen tief priorisiert. Es hat sich aber gezeigt, dass bereits am Anfang der Umsetzungs Phase viele Refactorings durchgeführt werden müssen. Dies nicht zuletzt aufgrund des schwer lesbaren Legacy Codes. Durch die Refactorings konnte die Verständlichkeit des Codes enorm verbessert werden. Dies spiegelt sich auch in den erhobenen Software Metriken wieder. Im folgenden Abschnitt sind diese genauer beschrieben.

5.2.1 Wartbarkeit

In der Wartbarkeits-Metrik [27] ist klar zu erkennen, dass die Wartbarkeit der meisten Klassen gut ist. Auf der X-Achse ist der Churn [5] abgebildet, also wie oft eine Klasse verändert wurde. Klassen die häufig angepasst werden, erledigen potentiell zu viele Aufgaben. Auf der Y-Achse ist die Wartbarkeit angezeigt. Diese errechnet sich aus Code-Duplikationen, zyklomatischer Komplexität von McCabe [28] und der Kognitiven Komplexität [6] einer Klasse. Die beiden Klassen *Persistor* und *ModelBuilder* weisen die schlechtesten Werte auf (oben rechts). Leider konnte aus Zeitgründen kein Refactoring der beiden Klassen durchgeführt werden. Ein entsprechender Hinweis ist aber im Ausblick 5.3.2 dokumentiert.

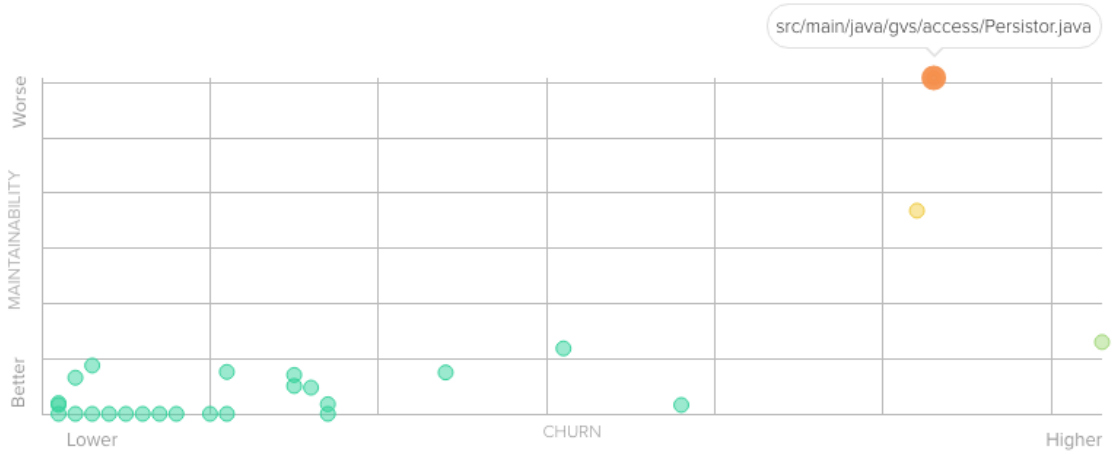


Abbildung 5.1 – Metrik: Wartbarkeit

5.2.2 Technische Schulden

Technischen Schulden (Abb. 5.2) beschreiben den zusätzlichen Aufwand, welcher für die Korrektur von schlecht geschriebener Software (Issues) aufgewendet werden muss. Das Verhältnis der Technischen Schulden (e.g. Technical Debt Ratio) errechnet sich wie folgt:

$$\text{Technical Debt Ratio} = \frac{\text{Remediation Time}_{total}}{\text{Implementation Time}_{total}}$$

Code Climate schätzt einerseits die benötigte Zeit um offene Issues zu beheben (*Remediation Time*) und andererseits die aufgewendete Implementations Zeit (*Implementation Time*) in Abhängigkeit zur Projektgrösse. Das Verhältnis dieser beiden Werte bildet das *Technical Debt Ratio*. Dieses ist im Idealfall so klein wie möglich. In der Grafik 5.2 ist sowohl die *Remediation Time* als auch das *Technical Debt Ratio* ersichtlich.

Das *Technical Debt Ratio* korreliert mit den Anzahl Zeilen an Source Code (Siehe 5.2.3). Auch gut zu erkennen ist, dass die technischen Schulden trotz weiteren Anpassungen am GVS nicht mehr zugenommen haben.

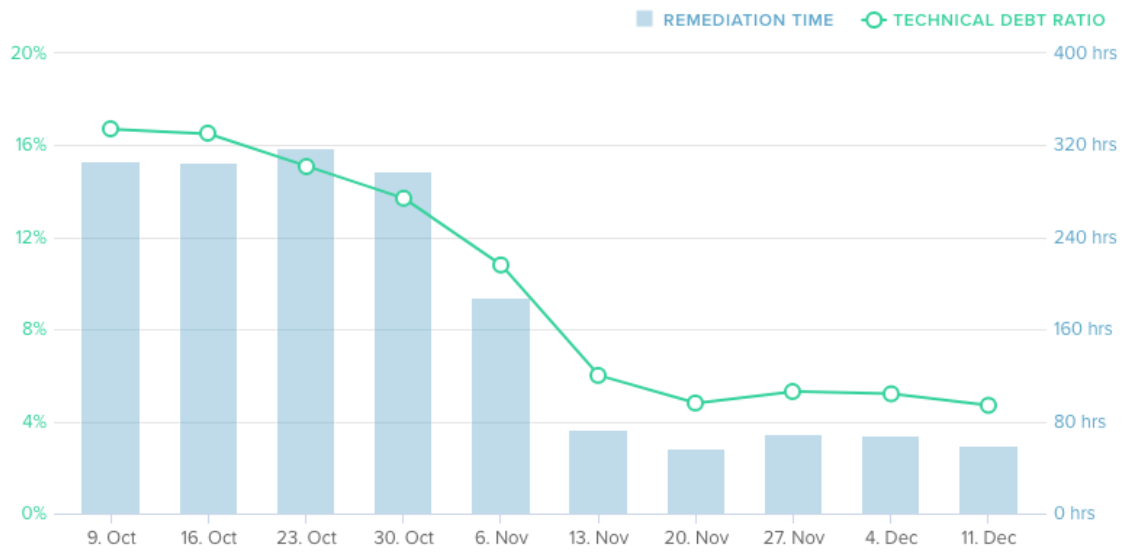


Abbildung 5.2 – Metrik: Technische Schulden

5.2.3 Lines of Code

Interessant zu beobachten ist, dass beinahe die selbe Funktionalität mit viel weniger Zeilen Code erreicht wurde. Dies ist insbesondere durch das Entfernen von dupliziertem Code möglich. Ein weiterer Einfluss hat der Einsatz von modernen Lambda Ausdrücken, die zur Zeit der Entwicklung von GVS 1.0 nicht zur Verfügung standen.

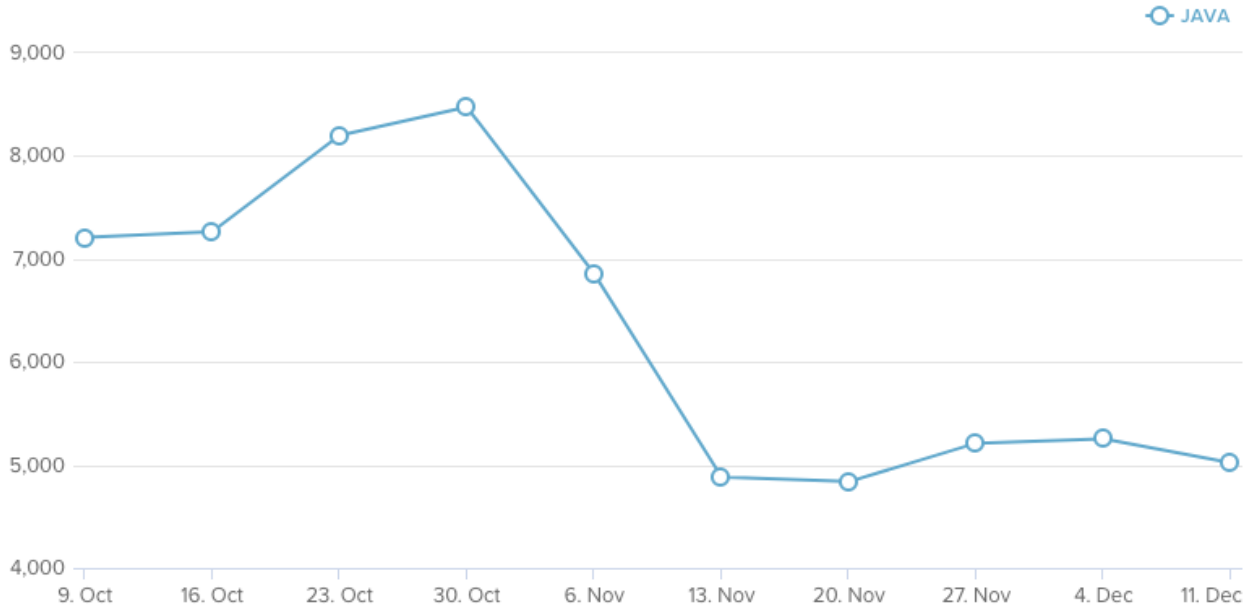


Abbildung 5.3 – Metrik: Lines of Code

5.2.4 Kritik

In der Analyse- und Entwurfsphase des Projekts wurde entschieden, dass die Hauptziele so umgesetzt werden sollen, dass möglichst viel bestehender Code wiederverwendet werden kann. Dieses Ziel wurde klar nicht erreicht.

Dies führte auch dazu, dass deutlich mehr Arbeitsaufwand für die Umsetzung des Projekts nötig war, als eingeplant wurde (siehe Anhang F). Die Wirtschaftlichkeit des Projekts hat dadurch gelitten.

Des Weiteren hat sich das Projektteam zu Beginn stark mit der Verbesserung der Architektur und Code Qualität beschäftigt. Dadurch gingen gewisse Feinheiten der Funktionalität etwas unter. Als Beispiel kann hier genannt werden, dass erst relativ spät klar wurde, wozu einige Einstellungen aus dem Kontextmenü - wie der Cluster Splitter oder das Soft-Layout - dienen.

5.3 Ausblick

Obwohl viel erreicht wurde im GVS 2.0 Projekt, konnten nicht alle Ideen umgesetzt und sämtliches Verbesserungspotential ausgeschöpft werden.

Dieses Kapitel bietet eine Übersicht, welche Aufgaben in einem nächsten GVS Major Update angegangen werden könnten.

5.3.1 Verbesserungspotential GVS 1.0

Während der Verwendung von GVS 1.0 sind einige Dinge aufgefallen, die nicht optimal funktionieren. Diese wurden vom Betreuer in einer Liste festgehalten und an das Projektteam überreicht. Gemäss der Besprechung vom 27.09.2017 wurden diese Änderungen mit tiefer Priorität in den Backlog eingepflegt. Dieses Kapitel zeigt auf, welche Punkte bis zum Abschluss des Projekts offen geblieben sind.

Font Size

GVS 3.0 sollte es dem User erlauben die Font Size der Labels zu verändern, ohne den Code von

GVS UI zu bearbeiten. Eignen würde sich entweder ein Menü-Eintrag "Einstellungen" oder das setzen der Font Size in der *GVSStyle* Klasse.

Unicode Labels

GVS 3.0 sollte die Darstellung von Unicode Zeichen in Labels unterstützen.

Snapshot Description

GVS 2.0 bietet einem User die Möglichkeit, eine Notiz zu einem Snapshot zu verfassen (siehe 4.10.2).

GVS 3.0 sollte das verwendete Protokoll und die GVS Lib Klassen so erweitern, dass solche Notizen direkt im Client erstellt und mit den Graphen mitgeschickt werden können.

Schnelle Beendigung der GVS Lib

Wenn die *main* Methode des Programms, welches GVS Lib verwendet sehr schnell beendet, gehen Daten bei der Übertragung verloren. Dies sollte bei GVS 3.0 verhindert werden.

5.3.2 Nicht umgesetzte Ideen im GVS 2.0

Refactoring ModelBuilder und Persistor

Die Klassen *ModelBuilder* und *Persistor* besitzen sehr ähnliche Funktionalität und somit viel *duplicated Code*. Obwohl diese Klassen in GVS 2.0 einem Refactoring unterzogen wurden, besteht das Problem weiterhin (siehe Abschnitt 5.2).

GVS 3.0 sollte diese Klassen soweit überarbeiten, dass das DRY Prinzip nicht mehr verletzt wird.

Automated Testing

GVS 1.0 besitzt keine automatisierten Tests. Auch in GVS 2.0 konnte diese Altlast nicht beseitigt werden. Es wurde nur eine sehr kleine Test Coverage erreicht, da die Erstellung von automatisierten Tests tief priorisiert wurde.

GVS 3.0 sollte den Business Layer und Teile des Access Layers sinnvoll mit Tests abdecken.

Anhang

Anhang A

Projektplanung

Datum	Version	Änderungen	Autor
25.09.17	1.0	Dokument erstellt	mwieland, mtrentini
28.09.17	1.1	Risikomanagement hinzugefügt	mtrentini
03.10.17	1.2	IDE, Frameworks, Buildprozess, Qualitätsmanagement hinzugefügt,	mwieland
05.10.17	1.3	Meilensteine definiert	mtrentini
11.12.17	1.4	Verantwortlichkeiten festgehalten	mtrentini

Tabelle A.1 – Versionshistory Projektplanung

A.1 Meilensteine

Meilenstein 1: Analyse

Analyse des Ist-Zustandes von GVS 1.0: Die Problemdomäne wird spezifiziert. Mehrere Diagramme zeigen die Schnittstelle zwischen dem Business Layer und dem Presentation Layer auf. Es ist ersichtlich, wie die einzelnen Komponenten miteinander kommunizieren bzw. wie die Daten von der Socket ans UI weiter gereicht werden.

Die in der Aufgabenstellung (D) geforderten Requirements werden festgehalten.

Meilenstein 2: Entwurf

Entwurf des Soll-Zustand von GVS 2.0: Eine klassische 3-Schichten Architektur wird entworfen. Es wird spezifiziert, welche GVS 1.0 Komponenten durch welche GVS 2.0 Komponenten ersetzt werden.

Meilenstein 3: Release 1

Der Presentation Layer ist vollständig gemäss Punkt 1 im Refactroing Konzept (3.3) umgesetzt.

Meilenstein 4: Release 2

Die im Refactoring Konzept (3.3) beschlossenen Änderungen sind umgesetzt. Das mit dem Enterprise Architekt erstellte Klassendiagramm ist mit dem Stand der GVS 2.0 Software synchron.

Meilenstein 5: Release 3

Alle im Meilenstein 2 definierten Requirements sind umgesetzt. Die GVS 2.0 Software ist für den Einsatz im Unterricht bereit.

#	Name	Artefakte	Datum
1	Analyse	Anforderungsspezifikation Klassendiagramm 1.0 Sequenzdiagramme Systemübersichts-Diagramm	11.10.17
2	Entwurf	Designspezifikation Klassendiagramm 2.0 Migrationstabelle	25.10.17
3	Release 1	GVS 2.0.1	22.11.2017
4	Release 2	GVS 2.0.2	06.12.2017
5	Release 3	GVS 2.0.3	22.12.2017

Tabelle A.2 – Meilenstein Planung

A.1.1 Artefact Overview

Im Artefact Overview sind sämtliche zu erbringende Arbeitsergebnisse mit ihren Fertigstellungsdaten aufgelistet. Wenn nötig sind verschiedene Versionen gekennzeichnet.

Artefact Overview

Milestone 1	
Milestone 2	
Milestone 3	
Milestone 4	
Milestone 5	

Meetingsprotokolle	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Plakat														x
Installationsanleitung														x
Benutzerhandbuch														x
Abstract														x
Management Summary														x
Persönlicher Bericht														x
Eigenständigkeitserklärung														x
Vereinbarung über Urheber- und Nutzungsrechte														x
Zeitauswertung														x
Metriken														x
Release GVS UI										v1		v2		v3
Testprotokolle												x		x
Release GVS Lib												v1		
Projektplanung						v1		v2						
Designspezifikation						v1								
Anforderungsspezifikation				v1										
Systemübersichts-Diagramm				v1										
Wichtige Sequenzdiagramme				v1								v2		
Klassendiagramm				v1								v2		
Artefact Overview	v1	v2				v3								
Phase	**Inception**		**Elaboration**				**Construction**						**Transition**	
Datum	20.09.	27.09.	04.10.	11.10.	18.10.	25.10.	01.11.	08.11.	15.11.	22.11.	29.11.	06.12.	13.12.	20.12.
Woche	1	2	3	4	5	6	7	8	9	10	11	12	13	14

A.2 Zeitplanung

Das Projekt wird im Rahmen der Studienarbeit durchgeführt. Insgesamt stehen 14 Wochen zur Verfügung in welchen jedes Teammitglied 240 Stunden leisten muss. Somit entstehen ca. 17 Stunden Arbeitsaufwand pro Woche und Teammitglied.

A.3 Projektverwaltung

Als Projektmanagement Software wird Jira [23] eingesetzt. In Jira werden alle Requirements als Issues erfasst und in den Product Backlog eingepflegt. Pro Iteration sollen jeweils so viele Issues eingeplant werden, wie unter Berücksichtigung von administrativen Aufgaben abgearbeitet werden können. Dabei spielt der Teamspeed eine grosse Rolle, welcher sich über die Projektdauer einpendeln soll.

A.3.1 Iterationsplanung

Die Iterationsplanung orientiert sich grob am Rational Unified Process (RUP) und ist unterteilt in eine Inception-, Elaboration-, Construction- sowie eine Transition-Phase. Jede Phase besteht aus einer oder mehreren Iterationen, die jeweils 1 Wochen dauern und am Donnerstag enden. Die Sprints sind bewusst nur 1 Woche lang, um eventuell ändernden Anforderungen gerecht zu werden. Zudem bringen die Wochenmeetings (A.3.4) jeweils neuen Input, welcher dann direkt in den nächsten Sprint einfliessen kann.

Eine Iteration wird nach SCRUM organisiert. Am Anfang des Sprints definiert das Projektteam die zu erledigenden Issues und schätzt deren Aufwand. (Siehe A.3.2) Jeweils am Ende eines Sprints wird die geleistete Arbeit reflektiert. Entsprechende Erkenntnisse und Verbesserungen werden fortlaufend in die Projektorganisation aufgenommen.

A.3.2 Schätzungen

Issues werden auf Basis von Story Points geschätzt. Dieses Vorgehen hat sich mit SCRUM etabliert. Die Nutzung von Story Points führt dazu, dass nicht die individuell unterschiedliche Bearbeitungszeit,

sondern die Komplexität einer User Story geschätzt wird. Dies vereinfacht und homogenisiert die Schätzungen und hilft den Teamspeed zu bestimmen.[34, 36]

A.3.3 Zeitauswertung

Für die Zeitauswertung wird das Jira Plugin Tempo [1] verwendet. Dieses bietet umfassende Auswertungsmöglichkeiten, sowie Exports nach MS Excel. Die Reports zur geleisteten Zeit befinden sich im Anhang F.

A.3.4 Meetings

Über die gesamte Projektdauer findet jeweils am Mittwoch um 17:15 ein wöchentliches Standortmeeting statt. Die Beschlüsse aus den Meetings werden protokolliert und bis spätestens 24h später an alle Teilnehmer versendet. Allfälliges Feedback wird nachträglich eingepflegt und versioniert abgelegt.

A.3.5 Branch Konzept

Die Branch Struktur orientiert sich an Gitflow [16]. Pro Issue wird ein Feature Branch erstellt, der nach erfolgreichem Review in den Development Branch gemerged wird. Beim Erreichen eines Meilensteins (siehe A.1) wird der Entwicklungszweig in den Master Branch gemerged.

A.4 Entwicklungsumgebung

Als IDE (Integrated Development Environment) wird Eclipse mit folgenden Plugins verwendet

- Buildship [11] für die Gradle Integration
- EGit [12] für die Git Integration
- ECLEmma [10] zur Überprüfung der Testabdeckung
- FindBugs [14] zur statischen Code Analyse
- Stan4J [33] zur Überprüfung von zyklischen Abhängigkeiten

- Checkstyle [3] zur Überprüfung der Coding Richtlinien.

A.5 Frameworks

A.5.1 Gluon Ignite mit Google Guice

Gluon Ignite [18] erweitert das verbreitete Dependency Injection-Framework Guice [20] für den Einsatz in JavaFX Applikationen. Mit Gluon Ignite lässt sich Dependency Injection nicht nur im Business Layer, sondern auch für die FXML Loader verwenden. Dependency Injection verringert die Kopplung zwischen Komponenten, verbessert die Testbarkeit und zentralisiert die Erzeugungslogik von Klassenobjekten.

A.5.2 ControlsFX

ControlsFX [7] ist ein Open Source Projekt für JavaFX UI Controls. Es erweitert die Standard-Controls um nützliche Elemente. Im GVS 2.0 wird es für den Toggle Button verwendet.

A.5.3 dom4j

Dom4j [8] ist ein XML Framework für Java. Es wird für das Schreiben und Lesen der XML Files verwendet, die über die Socket Verbindung gesendet werden.

A.5.4 SLF4J und Logback

SLF4J [32] und Logback [26] werden für das Schreiben von Log Einträgen in ein File sowie auf die Standardausgabe (STDOUT) verwendet.

A.6 Repositories

Die Artefakte werden in verschiedenen Repositories auf GitHub versioniert abgelegt. So gibt es für die Dokumentation, die Meeting-Protokolle und den Sourcecode jeweils ein eigenes Repository.

Repository	Nutzen	URL
gvs-ui	UI und Socket Server	https://github.com/Graphs-Visualization-Service/gvs-ui
gvs-lib-java	Library für Java Programme	https://github.com/Graphs-Visualization-Service/gvs-lib-java
gvs-lib-csharp	Library für C# Programme	https://github.com/Graphs-Visualization-Service/gvs-lib-csharp
gvs-tester	Ausführbare End-to-End Testfiles in Java	https://github.com/Graphs-Visualization-Service/gvs-tester

Tabelle A.3 – GVS 2.0 Repositories

A.7 Continuous Integration

Die Software wird nach jedem Commit mit Travis CI [37] gebuildet. Dies garantiert, dass sämtliche Qualitätsmassnahmen stets eingehalten werden. (Siehe A.8)

A.7.1 Buildprozess

Zur Erstellung der Software Artefakte wird Gradle [21] eingesetzt. Gradle verwaltet alle externen Software Abhängigkeiten. Zusätzlich wird die Software nur dann gebuildet, wenn alle Metriktools grünes Licht geben. Es resultiert ein ausführbares JAR (Java Archive).

A.8 Qualitätsmanagement

A.8.1 Unit und System Tests

Zur Sicherung der Software Qualität sind gute Tests unerlässlich. Da jedoch ein grosser Teil des Business Layers von GVS 1.0 übernommen wird, ist es nicht realistisch, für sämtliche Klassen Unit Tests nachzureichen. Wo immer möglich und sinnvoll sollen automatisierte Tests geschrieben werden. Der Schwerpunkt liegt dabei auf der Qualität und nicht auf einer möglichst hohen Testabdeckung.

Des Weiteren ist ein grosser Teil der Applikation an die Benutzeroberfläche gekoppelt. Diese Teile können nur schwer mit automatisieren Tests abgedeckt werden. Deshalb soll mit End-zu-End Tests der gesamte Funktionsumfang getestet und protokolliert werden (siehe Anhang E).

A.8.2 Definition of Done

Source Code wird erst zum Review freigegeben, wenn folgende Kriterien erfüllt sind.

- Die IDE rsp. Build Tool zeigt keine Warnungen und Fehler
- Es gibt keinen auskommentierten Code
- Es existieren sinnvolle Unit und Integrationstests für das Feature
- Alle Metriken geben grünes Licht
- Das Issue wurde in Jira [23] zum Review vermerkt

A.8.3 Review

Nach Abschluss eines Issues wird dieses an den Teampartner zum Review übergeben. Durch das Vier-Augen-Prinzip kann die Qualität des Produktes hoch gehalten werden. Zusätzlich haben beide Teampartner stets den selben Wissensstand.

A.8.4 Metriken

Mit den Metriken kann während der Entwicklung geprüft werden, ob die Code Qualität den gesetzten Vorgaben (Siehe A.8.2) entspricht.

Code Style

Als Styleguide wird eine angepasste Version des Sun Styleguide verwendet [4]. Damit dieser im Projekt durchgängig eingehalten wird, wird das Checkstyle Plugin [3] in den Build-Prozess integriert. (Siehe A.7.1)

Code Climate

Zur Überwachung des Technical Debt und der Wartbarkeit wird die Github App Code Climate [22] eingesetzt.

A.8.5 Refactorings

Aufgrund der erkannten Mängel in der Analysephase (Siehe 2.2.6) werden fortlaufend Refactorings des migrierten Codes durchgeführt.

A.9 Risikomanagement

Nachfolgend sind die wichtigsten Risiken für diese Arbeit aufgelistet. Gewichtet werden die Risiken nach Eintrittswahrscheinlichkeit und Schadenshöhe. [31] Auf eine Schätzung bezüglich dem zeitlichen Mehraufwand wird bewusst verzichtet, da diese erfahrungsgemäss sehr ungenau ist und wenig praktischen Nutzen hat. Dennoch erzwingt die Liste ein aktives Auseinandersetzen mit den möglichen Risiken und zeigt Wege für die Eskalation rsp. Reaktion beim Eintritt eines Risikos.

#	Eintrittswahrscheinlichkeit (E)	Schadensschwere (S)
1	gering	leicht
2	mittel	mittelschwer
3	hoch	schwer

Tabelle A.4 – Legende Risiken

A.9.1 Mögliche Risiken

#	Beschreibung	E	S	Prävention	Massnahmen bei Eintritt
1	Verlust von Daten auf Grund von technischen Störungen oder Diebstahl von persönlichen Notebooks	2	1	Backups (siehe A.9.2)	Durch die Massnahmen muss höchstens ein Datenverlust von 8h aufgearbeitet werden.
2	Die Schnittstelle zwischen Business Logik und Presentation Layer ist weitaus weniger wohldefiniert als ursprünglich angenommen. Der nötige Zeitaufwand übersteigt die Schätzung um ein Vielfaches	1	2	Gute Recherche und Einarbeit in GVS 1.0, genügend lange Elaboration Phase	Rücksprache mit Betreuer. Überprüfen des Projektscopes und allenfalls Anpassung des Scopes.
3	Funktionalitäten von GVS 1.0, welche mit Swing umgesetzt wurden, werden durch JavaFX nicht oder unzureichend unterstützt	1	2	Einlesen in JavaFX [13], Funktionalitäten GVS 1.0 abklären	Alternative zur bestehenden Umsetzung finden
4	Eingesetzte Frameworks, Libraries und Cloud Services harmonieren nicht wie angenommen	2	2	Auf einen Software Stack setzen, der beliebt und weit verbreitet ist. Dies reduziert das Risiko, dass der gegenseitige Support fehlt.	Alternativen für inkompatible Services finden
5	Der geplante Funktionsumfang von GVS 2.0 übersteigt das Zeitbudget, dass im Rahmen der Studienarbeit zur Verfügung steht	2	3	Endprodukt in Teilkomponenten aufteilen. Pro Komponente den zeitlichen Aufwand bewerten. Klare Definition, welche Artefakte zwingend umgesetzt werden müssen.	Frühzeitige Rücksprache mit dem Betreuer, wenn sich abzeichnet, dass eine Komponente mehr Arbeit in Anspruch nimmt.

Tabelle A.5 – mögliche Risiken

A.9.2 Backups

Zur Minimierung von allfälligen Datenverlusten wird wie folgt vorgegangen:

1. Täglich automatisiertes Backup aller Projektdaten im JIRA [23] (Zeiterfassung, erstellte Issues, JIRA Konfiguration)
2. Die Projektdokumentation sowie der Programmcode wird in den vier Github Repositories der Github Organisation [17] versioniert abgelegt. (Siehe Tabelle A.3) Für die Projektmitglieder gilt der Grundsatz "Commit early and often".
3. Weitere Artefakte werden auf Dropbox [9] abgelegt und dort automatisch gesichert und versioniert.

A.10 Verantwortlichkeiten

Um Unklarheiten und Unstimmigkeiten zu vermeiden sind folgende Verantwortlichkeiten zugeteilt:

Bereich	Verantwortliche/r
Analyse	beide
Design	beide
Testing (Usability, Systemtests)	mtrentini
Projektmanagement (inkl. Protokolle)	beide
Entwicklungsumgebung, CI	mwieland

Tabelle A.6 – Verantwortlichkeiten

Beide Team-Mitglieder arbeiten jedoch in allen Bereichen des Projekts mit. Die genaue Arbeitsteilung ist der git-History, den Java-Docs sowie den Änderungshistories der Projektdokumentation zu entnehmen.

Anhang B

Klassendiagramme

Die folgenden Seiten enthalten die Entwicklung der Klassendiagramme über den Projektverlauf. Für die GVS Lib Komponente existierte in der Vorgängerarbeit kein Klassendiagramm.

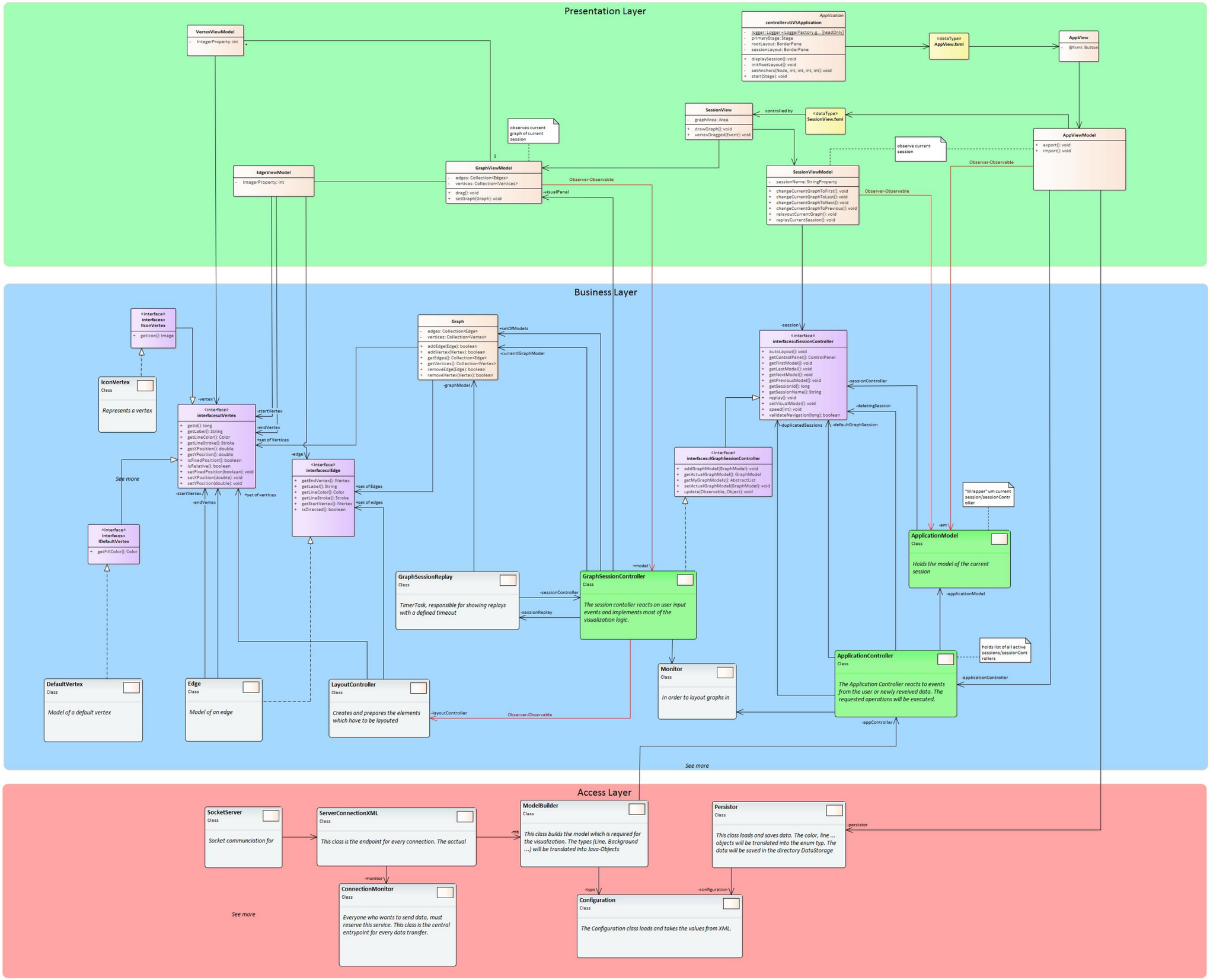
1. GVS 1.0 Klassendiagramm UI Analyse
2. GVS 2.0 Klassendiagramm UI Entwurf
3. GVS 2.0 Klassendiagramm UI Umsetzung
4. GVS 2.0 Klassendiagramm Lib Umsetzung

class gvs

GVS 1.0 Klassendiagramm UI Analyse

class gvs2.1

GVS 2.0 Klassendiagramm UI Entwurf



class Klassendiagramm

GVS 2.0 Klassendiagramm UI Umsetzung

UI

AppView
Class
MVVM View Class. Defines functions for all fxml actions. Defines bindings to current session and active sessions.

-sessionViewController

SessionView
Class
MVVM View Class.

EdgeViewModel
Class
Contains JavaFX Properties which are used for bidirectional bindings. See more

-appViewModel

AppViewModel
Class
The ViewModel class for the GVS Application. Corresponds to the classical ViewModel of the MVVM Pattern. It observes the ApplicationModel See more

-sessionViewModel

SessionViewModel
Class
The ViewModel class for the current session. Corresponds to the classical ViewModel of the MVVM Pattern. It observes the ApplicationModel and handles See more

-graphViewModel

GraphViewModel
Class
Represents one snapshot of a graph visualization. Observes the CurrentGraphHolder.

-endVertex -startVertex

VertexViewModel
Class
Contains JavaFX Properties which are used for bidirectional bindings. See more

Business

-appController

ApplicationController
Class
The Application Controller reacts on events from the user or newly See more

ILayouter
Interface
-graphLayouter -treeLayouter

TreeLayouter
Class
Layouts a tree according to the Reingold-Tilford Algorithm which was improved by Walker et al. to accomodate for n-ary trees See more

GraphLayouter
Class
Creates and prepares the elements which need to be layouted.
Executes a separate ticker thread which calls the See more

SessionReplay
Class
TimerTask, responsible for showing replays with a defined timeout

-currentReplayThread

Model

-graphHolder

GraphHolder
Class
Holds the current graph. State class See more

-sessionPlaceHolder -currentSession -session

Session
Class
A session represents a collection of graphs. It contains logic to navigate through its graphs, which included the replay functionality.

-currentGraph -graphs 0..* -referenceGraph

Graph
Class
Represents a graph

-sessions 0..* -currentSession

-sessionHolder -sessionHolder

SessionHolder
Class
Holds the model of the current session.

-sessionHolder

-sessionType

ISessionType
Interface
Interface to label See more

TreeSessionType
Class
Marks a session as a tree See more

GraphSessionType
Class
Marks a session as a graph See more

-vertex -startVertex -endVertex

IVertex
Interface

IEdge
Interface
-edge

GraphVertex
Class
Model of a default vertex See more

TreeVertex
Class
Model of a tree vertex. It represents See more

LeafVertex
Class
This class is only See more

-style -style -style

GVSStyle
Class

Edge
Class
Model of an edge See more

Access

-persistor

Persistor
Class
Loads and saves sessions. While loading, it translates XML into See more

ModelBuilder
Class
This class builds the model which is required for the visualization. The See more

-modelBuilder

-graphSessionFactory -sessionFactory

SessionFactory
Interface
Factory interface for Guice See more

SocketServer
Class
Socket Endpoint of GVS UI. The server is running in its See more

ClientConnection
Class
This class is the endpoint for each incoming See more

-watchdog

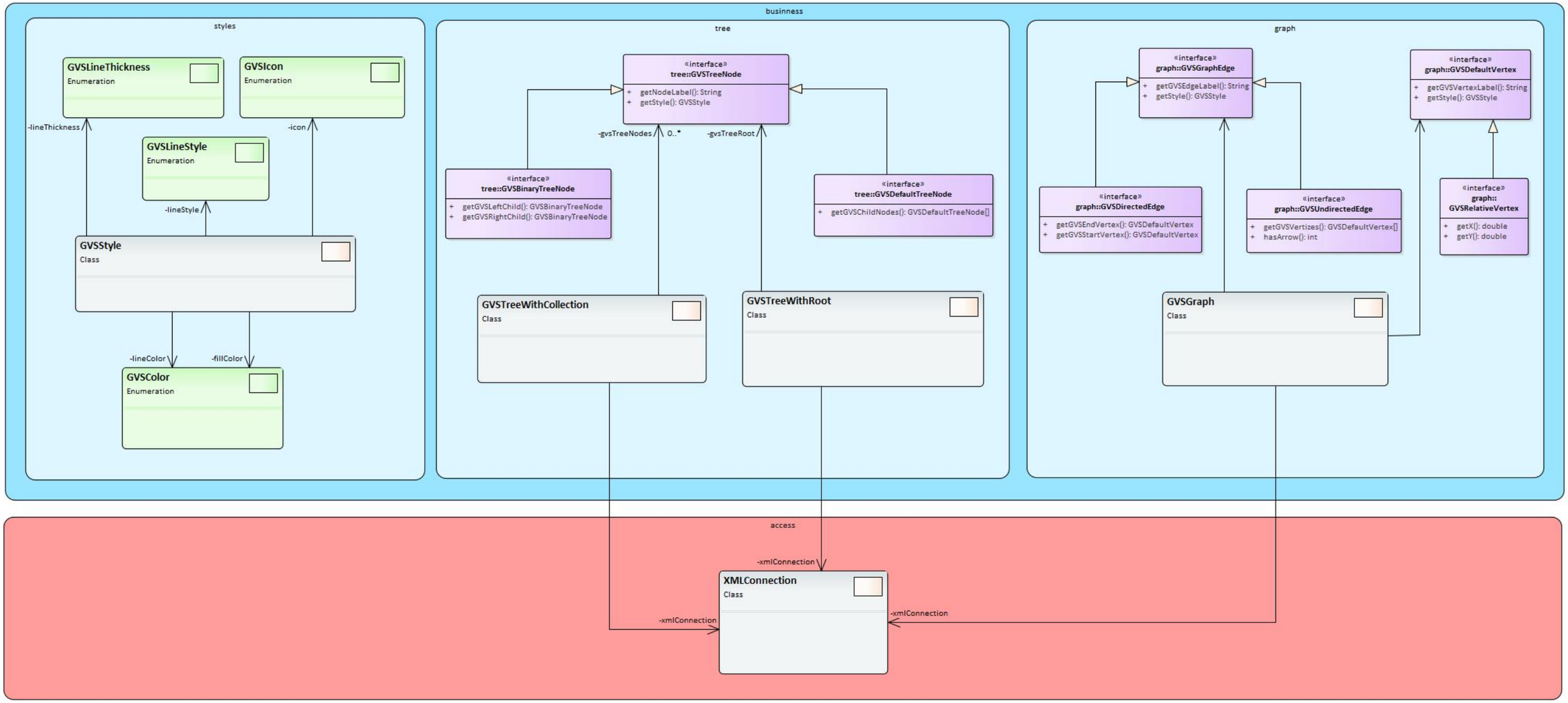
ConnectionMonitor
Class
Guarantees that only one client can See more

-monitor -monitor

Watchdog
Class
The Watchdog is responsible for See more

class Klassendiagramm

GVS 2.0 Klassendiagramm Lib Umsetzung



Anhang C

Migrationsliste

Die Migrationsliste gibt einen Überblick über die geplante Migration aller GVS 1.0 Klassen. Die angedachte Package Struktur entspricht nicht der effektiv umgesetzten, da sich diese in der Umsetzungsphase noch geändert hat.

GVS 1.0 Migration: Packages and classes

Legende	
Entfernen	
Behalten	
Neu	
Weiter untersuchen	

GVS 1.0: Package	GVS 1.0 Class	GVS 2.0 Package	GVS 2.0 Class	Bemerkung 1.0	Bemerkung 2.0	Weitere Schritte
gvs.visualization.application.view	CheckableItem	-	-		Neu als FXML	
gvs.visualization.application.view	CheckListRenderer	-	-		Neu als FXML	
gvs.visualization.application.view	ComboItem	-	-		Neu als FXML	
gvs.visualization.application.view	ComboListener	-	-		Neu als FXML	
gvs.visualization.application.view	ComboRenderer	-	-		Neu als FXML	
gvs.visualization.application.view	RadioButttonUtils	-	-	Wird in GVS1.0 gar nie verwendet	Neu als FXML	
gvs.visualization.application.view	SaveDialog	-	-		Neu als FXML	
gvs	GVSLauncher	gvs	GVSApplication		Startet die Applikation und initialisiert die FXML Files	
gvs.common	Configuration	gvs.access				
gvs.server	ConnectionMonitor	gvs.access			Dient dem Locking (Race Conditions)	
gvs.server	ModelBuilder	gvs.access				ModelBuilder und Persistor in eine Klasse mergen
gvs.common	Persistor	gvs.access		Ähnliche Funktionalität wie im Model Builder		ModelBuilder und Persistor in eine Klasse mergen
gvs.server.socket	ServerConnectionXML	gvs.access				
gvs.server.socket	SocketServer	gvs.access				
gvs.visualization.application.controller	ApplicationController	gvs.business.logic				
gvs.visualization.graph.controller	GraphSessionController	gvs.business.logic		Observer of LayoutController		CurrentSession in eigene Model Klasse herausziehen
gvs.visualization.graph.controller	GraphSessionReplay	gvs.business.logic		responsible for showing replays with a defined timeout		
gvs.visualization.graph.layout	LayoutController	gvs.business.logic	GraphLayoutController			
gvs.visualization.graph.layout	LayoutGuard	gvs.business.logic				
gvs.visualization.application.controller	Monitor	gvs.business.logic		Lockt während dem Layouting Vorgang		
gvs.visualization.graph.layout.ticker	Tickable	gvs.business.logic				
gvs.visualization.tree.layout	TreeLayoutController	gvs.business.logic				
gvs.visualization.tree.controller	TreeSessionController	gvs.business.logic				
gvs.visualization.tree.controller	TreeSessionReplay	gvs.business.logic				
gvs.visualization.tree.view	ClusterSplitter	gvs.business.logic.cluster		ClusterSplitter		Fehlerbehebung
gvs.visualization.tree.view	ClusterSplitterGVS	gvs.business.logic.cluster		ClusterSplitter		Fehlerbehebung
gvs.visualization.tree.view	NodeClusterImpl	gvs.business.logic.cluster		ClusterSplitter		Fehlerbehebung
gvs.visualization.tree.view	NodeLineImpl	gvs.business.logic.cluster		ClusterSplitter		Fehlerbehebung
gvs.visualization.graph.layout.helpers	Area	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.helpers	AreaDimension	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.helpers	AreaPoint	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.ticker	AreaTicker	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.helpers	AreaVector	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.ticker	HitsPerSecond	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.helpers	Particle	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.rules	RepulsiveForce	gvs.business.logic.physics		Physics Engine		
gvs.visualization.graph.layout.rules	Traction	gvs.business.logic.physics		Physics Engine		
gvs.visualization.application.model	ApplicationModel	gvs.business.model		Holds the model of the current session Nur Referenz auf SessionController und Observable Impl.	Zugriffspunkt für Observer: ApplicationView	merge with ApplicationController
gvs.visualization.graph.model	DefaultVertex	gvs.business.model.graph				
gvs.visualization.graph.model	Edge	gvs.business.model.graph				
gvs.visualization.graph.model	GraphModel	gvs.business.model.graph	Graph			
gvs.visualization.graph.model	IconVertex	gvs.business.model.graph				
gvs.visualization.tree.model	BinaryNode	gvs.business.model.tree				
gvs.visualization.tree.model	DefaultNode	gvs.business.model.tree				
gvs.visualization.tree.model	TreeModel	gvs.business.model.tree	Tree			
gvs.interfaces	IBinaryNode	gvs.interfaces				
gvs.interfaces	IDefaultNode	gvs.interfaces				
gvs.interfaces	IDefaultVertex	gvs.interfaces				
gvs.interfaces	IDefaultVertexComponent	gvs.interfaces			JavaFX Model mit Properties zum binden an die View, Observer von 'BusinessModel'	bei Refactoring löschen
gvs.interfaces	IEdge	gvs.interfaces				Style Komponenten in externe Style Klasse extrahieren
gvs.interfaces	IGraphSessionController	gvs.interfaces				allenfalls bei Refactoring löschen (unnötige Vererbung)
gvs.interfaces	IIconVertex	gvs.interfaces				
gvs.interfaces	IIconVertexComponent	gvs.interfaces			JavaFX Model mit Properties zum binden an die View, Observer von 'BusinessModel'	bei Refactoring löschen
gvs.interfaces	INode	gvs.interfaces				Style Komponenten in externe Style Klasse extrahieren
gvs.interfaces	ISessionController	gvs.interfaces				
gvs.interfaces	ITreeSessionController	gvs.interfaces				allenfalls bei Refactoring löschen (unnötige Vererbung)
gvs.interfaces	IVertex	gvs.interfaces				Style Komponenten in externe Style Klasse extrahieren
gvs.interfaces	IVertexComponent	gvs.interfaces	IVertexViewModel		JavaFX Model mit Properties zum binden an die View, Observer von 'BusinessModel'	
gvs.interfaces	IVisualizationGraphPanel	gvs.interfaces				
gvs.interfaces	IVisualizationPanel	gvs.interfaces				
gvs.interfaces	IVisualizationTreePanel	gvs.interfaces				
gvs.visualization.application.view	ApplicationView	gvs.ui.logic.app+gvs.ui.view.app	AppViewModel + AppView.fxml	JFrame, Observer of ApplicationModel	Wird neu als FXML dargestellt.	
gvs.visualization.graph.view	DefaultVertexComponent	gvs.ui.model.graph	DefaultVertexViewModel		JavaFX Model mit Properties zum binden an die View, Observer von 'BusinessModel'	
gvs.visualization.graph.view	EdgeComponent	gvs.ui.model.graph	EdgeViewModel		JavaFX Model mit Properties zum binden an die View, Observer von 'BusinessModel'	
gvs.visualization.graph.view	IconVertexComponent	gvs.ui.model.graph	IconVertexViewModel		JavaFX Model mit Properties zum binden an die View, Observer von 'BusinessModel'	
gvs.visualization.graph.model	VisualizationGraphModel	gvs.ui.model.graph	GraphViewModel	gives updates to the visualization panel		
gvs.visualization.tree.view	DefaultNodeComponent	gvs.ui.model.tree	NodeViewModel		JavaFX Model mit Properties zum binden an die View, Observer von 'BusinessModel'	
gvs.visualization.tree.model	VisualizationTreeModel	gvs.ui.model.tree	TreeViewModel	gives updates to the visualization panel		in TreeModel bzw neu Tree mergen
gvs.visualization.application.view	ControlPanel	gvs.ui.view	SessionViewModel + SessionView + Session.fxml	JPanel, Zeichnet die Buttons für prev,next,...		
gvs.visualization.application.view	ApplicationAction	gvs.ui.view.app	AppView + AppView.fxml	Represents most of the application and menu actions. (Load, Save, Exit, ..)	Beinhaltet alle Event Listener	
gvs.visualization.graph.view	VisualizationGraphPanel	gvs.ui.view.graph + gvs.ui.logic.graph	GraphViewModel + GraphView			
gvs.visualization.tree.view	VisualizationTreePanel	gvs.ui.view.tree + gvs.ui.logic.tree	TreeViewModel + TreeView			
gvs.visualization.graph.view	LabelConflictCheck			Checks for conflicts between edge labels. Used by VisualizationGraphPanel	Replaced by JavaFX Logic	
		gvs.util	Color	values siehe config.xml	Enum für Colors [lightGray, blue, brown...]	
		gvs.util	LineStyle	values siehe config.xml	Enum [standard, dotted, through...]	
		gvs.util	LineThickness	values siehe config.xml	Enum [standard, bold, slight...]	

Anhang D

Aufgabenstellung

Die folgenden Seiten enthalten die offizielle Aufgabenstellung dieser Studienarbeit.

Graphs-Visualization-Service GVS 2.0

Studenten

- Murièle Trentini
- Michael Wieland

Einführung

Bei den Modulen "Algorithmen und Datenstrukturen 1" und "Algorithmen und Datenstrukturen 2" werden Algorithmen zur Bearbeitung von Bäumen und allgemeinen Graphen programmiert.
Dabei ist die Programmierung für die Visualisierung derartiger Algorithmen und den dazugehörigen Datenstrukturen mit relativ grossem Aufwand verbunden und nicht Bestandteil des Modul-Stoffes.

Aus diesem Grund wurde dazu im Jahre 2005 der 'Graphs-Visualization-Service' erstellt. Dieser Service ist für den Einsatz im Unterricht ausgelegt um die Studenten bei der Programmierung zu unterstützen.
Er besteht serverseitige aus einer Java-Applikation und clientseitig steht je ein API für Java und .NET zur Verfügung.

Mit einem Major-Upgrade soll das Tool nun auf die aktuellen Technologien aktualisiert werden.

Aufgabenstellung

Beim aktuellen 'Graphs-Visualization-Service' soll unter Berücksichtigung resp. Beibehaltung der aktuellen Funktionalität:

- serverseitig Swing durch JavaFX ersetzt werden
- clientseitig das API um Generics erweitert werden
- punktuelle Verbesserungen vorgenommen werden

Unter Berücksichtigung von aktuellen Software-Engineering-Methoden soll ein geeigneter Entwicklungsprozess definiert und darauf basierend das Projekt realisiert werden.

Technologien

- Java
- Enterprise Architect

Generelles

- Die Vorgaben der Abteilung Informatik [1] sind einzuhalten, insbesondere die Anleitung zur Dokumentation [2].
- Die "Generelle Richtlinien für Studien- und Bachelorarbeiten" [3] sind einzuhalten.
- Mit dem CASE-Tool Enterprise Architect ist ein UML-Modell zu führen, welches synchron mit den Programm-Sourcen und der Projekt-Dokumentation ist.
- Ein Java-Entwickler muss mit der Projekt-Dokumentation in die Lage versetzt werden, die Applikation in Betrieb zu nehmen und weiter entwickeln zu können.

Termine

- Montag, 18.09.17 Beginn der Studienarbeit
- Freitag, 22.12.17 17:00 Uhr Abgabe der Studienarbeit

Betreuung

- Betreuer
  Thomas Letsch
  tletsch@hsr.ch
  055 - 22 24 567 (HSR Büro 5.204); 055 - 214 43 50 (Geschäft)
- Besprechungen
  Wöchentliche Besprechung jeweils Mittwoch 17:15 Uhr

Referenzen

[1] Skripte-Server: /Informatik/Fachbereich/Studienarbeit_Informatik/SAI
[2] Anleitung Dokumentation BA_SA_170905.pdf
[3] "Generelle Richtlinien für Studien- und Bachelorarbeiten"
(v1.7 / 19.08.2015, Thomas Letsch)

Rapperswil, 18.September 2017

Thomas Letsch

Anhang E

Testprotokoll

E.1 Systemtests

Die folgenden Seiten enthalten die Testprotokolle der beiden Systemtest-Durchführungen.

1. System Tests 04.12.17
2. System Tests 15.12.17

Systemtest: Graph - Normale Fälle

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.graph	-	Nicht Bestanden
Datum	04.12.17		

Testergebnis ✔75%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	RelativeGraph.java	Darstellung eines Graphen mit relativen Vertices, deren Positionen vom User fixiert sind.	Koordinaten der Vertices werden prozentual interpretiert.	+	
2	DefaultGraph.java	Darstellung eines Graphen mit default Vertices, deren Positionen vom User nicht angegeben werden.	Koordinaten der Vertices werden vom Layouter für den ersten Graphen berechnet. Für weitere Graphen der gleichen Session werden die Koordinaten übernommen.	+	
3	RelativeGraph.java DefaultGraph.java	Darstellung von gerichteten Edges.	Edge wird als Pfeil dargestellt. Der Schnittpunkt des Pfeils mit dem Vertex ist korrekt berechnet und verändert sich beim Dragging des Vertex entsprechend.	+	Es wäre allenfalls etwas schöner (v.a. bei kurzen Edges), wenn das Label mittig zur linie "ohne Pfei" wäre
4	RelativeGraph.java DefaultGraph.java	Darstellung von ungerichteten Edges ohne Pfeilangabe.	Edge wird als Linie dargestellt. Der Schnittpunkt der Linie mit dem Vertex ist korrekt berechnet und verändert sich beim Dragging des Vertex entsprechend.	+	
5	RelativeGraph.java DefaultGraph.java	Darstellung von ungerichteten Edges mit Pfeilangabe.	Edge wird als Pfeil dargestellt. Der Schnittpunkt des Pfeils mit dem Vertex ist korrekt berechnet und verändert sich beim Dragging des Vertex entsprechend.	+	
6	IconGraph.java	Darstellung eines Graphen mit Vertices, welche teilweise ein Label mit Icon und teilweise nur ein Icon enthalten.	Korrekte Darstellung des Graphen.	+	
7	RelativeGraph.java DefaultGraph.java	Darstellung einer Abfolge von Graphen mit Styleänderungen.	Korrekte Darstellung der Styles.	+	
8	RelativeGraph.java DefaultGraph.java	Darstellung einer Abfolge von Graphen mit Hinzufügen und Entfernen von Vertices.	Korrekte Darstellung des Graphen. Neu hinzugekommene Vertices werden gelayoutet. Gleichbleibende Vertices übernehmen die Positionen des vorherigen Graphen.	-	Bei DefaultGraphen wird der neue Vertex nicht per physics engine plaziert, sondern erscheint bei 0,0 Nach betätigen von Autolayout i.o.
9	RelativeGraph.java DefaultGraph.java	Abspeicherung einer Graph Session.	Die Session wird korrekt gespeichert. Der Speicherort wird vom User gewählt. Die korrekte Dateiendung (*.gvs) wird gesetzt. Die berechneten Koordinaten werden mitgespeichert.	+	
10	relativeGraph.gvs	Laden einer Graph Session	Die Session wird korrekt geladen. Es werden nur Dateien mit der *.gvs Endung angezeigt. Die gespeicherten Koordinaten werden übernommen. Die Graphen können mit AutoLayout neu gelayoutet werden.	+	

11	RelativeGraph.java DefaultGraph.java	Replay einer Graph Session	Die Session wird korrekt abgespielt. Während dem Abspielen wird der Play Button zum Pause Button. Nur der Pause und der Cancel Button sind während dem Abspielen aktiv. Graphen können von jedem Step aus abgespielt werden. Der Step wird beim Pausieren beibehalten. Beim Cancel springt die Session auf den ersten Graphen.	+	
12	RelativeGraph.java DefaultGraph.java	Durchsteppen eines Graphs	Die Step Buttons (First, Prev, Next, Last) navigieren korrekt durch die Session. Bei den ersten und letzten Graphen sind die korrekten Buttons aktiviert/deaktiviert. Die Progressbar zeigt den korrekten Füllstand.	+	
13	RelativeGraph.java DefaultGraph.java	DragSupport eines Graphen	Vertices können herum gezogen werden. Dabei bewegen sich die angehängten Edges entsprechend mit. Beim Dragen eines Vertex an den rechten oder unteren Rand, wird der gesamte Graph herausgezoomt um Platz zu schaffen.	+	
14	RelativeGraph.java DefaultGraph.java	AutoLayout eines Graphen	Die Koordinaten der Vertices werden neu berechnet. Bei aktivierter Random Layout Option werden die Vertices zufällig oder bei deaktivierter Option immer gleich "zufällig" positioniert. Während dem Layoutprozess sind sämmtliche anderen Controls deaktiviert.	-	Nach dem layouten werden die StepButtons nicht korrekt deaktiviert
15	StyleTester.java	Sämtliche unterstützten Styles für Edges und Nodes werden übertragen und dargestellt (Farben, Liniendicke, Linienstyle)	Alle Styles werden wie im Client definiert dargestellt	-	Dicke Linie (FAT) scheint nur für ArrowHeads und nicht für Linien zu funktionieren. Die Edge Label Farbe ändert teilweise nicht deterministisch auf Weiss.
16	-	Autolayout nachdem alle Vertices mit dem Mauszeiger manuell positioniert wurden	Eine entsprechende Meldung erscheint, dass alle Nodes bereits positioniert sind.	-	Führt zu inkorrekter aktiviertung der StepButtons

Systemtest: Graph - Spezielle Fälle

System under Test	https://github.com/Graphs-Visualization-Service
Package	gvs.tester.graph
Datum	04.12.17

\+ Bestanden
\- Nicht Bestanden

Testergebnis ✔80%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	LongLabels.java	Darstellung eines Graphen mit langen Vertex Labels.	Die langen Labels werden abgekürzt. Beispiel: LongLabel -> Lo....el	+	
2	mixedGraph.java	Darstellung eines Graphen der Default Vertices und relative Vertices enthält.	?	-	wird vom XML Schema nicht erkannt. Kein Absturz, nur Fehler in Console. User Feedback!
3	emptyGraph.java	Übermittlung eines Graphen der keine Vertices enthält.	Leerer Screen wird dargestellt. Name der Session erscheint im Dropdown Menü.	+	Braucht es hier eine UserInfo à la "empty Graph"?
4	nullEdges.java	Übermittlung eines Graphen der Edges enthält, welche null als Vertex-Start oder Ende besitzen.	Edges sollen nicht angezeigt werden. Es kommt zu keinen Fehlern im GVS 2.0 UI	+	
5	emptyString.java	Darstelllung eines Graphen mit Vertices, welche leere Strings als Labels besitzen.	Vertices werden trotzdem angezeigt. Sie besitzen kein Label.	+	Allenfalls könnte man eine minWidth einstellen, dass in so einem Fall schöne Kreise entstehen statt "Eier"

Systemtest: Tree - Normale Fälle

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.tree	-	Nicht Bestanden
Datum	04.12.17		

Testergebnis ✔92%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	BinaryTree.java	Dartellung eines Baums mit binären Nodes. Dabei sollen gewisse Nodes nur ein linkes oder nur ein rechtes Kind besitzen.	Platzierung der Nodes wird vom Layouter richtig berechnet. AutoLayout sowie Dragging ist deaktiviert.	+	
2	MultipleRoots.java	Darstellung eines Forests	Die einzelnen Bäume überlappen sich nicht.	+	
3	BinaryTree.java	Darstellung einer Abfolge von Trees mit Hinzufügen und Löschen von Nodes.	Korrekte Darstellung der Session	+	
4	BinaryTree.java	Darstellung einer Abfolge von Trees mit Styleänderungen.	Korrekte Darstellung der Styles. Edges übernehmen immer den Style des Kind Nodes.	+	
5	DefaultTree.java	N-ary Trees werden von GVS 2.0 Lib nicht unterstützt.	Exception im Client	-	Wenn noch Zeit ist könnten wir das ergänzen, Layoutalgorithmus sollte funktionieren. Exception ist eine Cast Exception. Besser wäre hier etwas das klar macht, dass der Fehler mangelnde Implementierung von DefaultTrees ist.
6	BinaryTree.java	Abspeicherung einer Tree Session.	Die Session wird korrekt gespeichert. Der Speicherort wird vom User gewählt. Die korrekte Dateiendung (*.gvs) wird gesetzt.	+	
7	test.gvs	Laden einer Tree Session	Die Session wird korrekt geladen. Es werden nur Dateien mit der *.gvs Endung angezeigt.	+	
8	BinaryTree.java	Replay einer Tree Session	Die Session wird korrekt abgespielt. Während dem Abspielen wird der Play Button zum Pause Button. Nur der Pause und der Cancel Button sind während dem Abspielen aktiv. Graphen können von jedem Step aus abgespielt werden. Der Step wird beim Pausieren beibehalten. Beim Cancel springt die Session auf den ersten Graphen.	+	
9	BinaryTree.java	Durchsteppen eines Trees	Die Step Buttons (First, Prev, Next, Last) navigieren korrekt durch die Session. Bei den ersten und letzten Graphen sind die korrekten Buttons aktiviert/deaktiviert. Die Progressbar zeigt den korrekten Füllstand.	+	
10	BinaryTree.java	DragSupport eines Trees	DragSupport für Trees ist deaktiviert.	+	
11	BinaryTree.java	AutoLayout eines Trees	AutoLayout für Trees ist deaktiviert.	+	
12	StyleTester.java	Sämtliche unterstützten Styles für Edges und Nodes werden übertragen und dargestellt (Farben, Liniendicke, Linienstyle)	Alle Styles werden wie im Client definiert dargestellt	+	
13	ClusterSplitter	Darstellung überfüllter Trees	Baum wird trotz vieler Nodes schön dargestellt.	+	

Systemtest: Tree - Spezielle Fälle

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.tree	-	Nicht Bestanden
Datum	04.12.17		

Testergebnis ✔100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	LongLabels.java	Darstellung eines Trees mit langen Vertex Labels.	Die langen Labels werden abgekürzt. Beispiel: LongLabel -> Lo....el	+	
2	EmptyString.java	Darstelllung eines Trees mit Vertices, welche leere Strings als Labels besitzen.	Nodes werden trotzdem angezeigt. Sie besitzen kein Label.	+	Allenfalls könnte man eine minWidth einstellen, dass in so einem Fall schöne Kreise entstehen statt "Eier"
3	RemoveRoot.java	Darstellung einer Tree Abfolge, wenn die Root entfernt wird. Dabei wird in der GVS 2.0 Lib die Struktur GVSTreeWithRoot benutzt.	Leerer Screen wird dargestellt. Name der Session erscheint im Dropdown Menü.	+	Braucht es hier eine UserInfo à la "empty Graph"?
4	RemoveRootCollection.java	Darstellung einer Tree Abfolge, wenn die Root entfernt wird. Dabei wird in der GVS 2.0 Lib die Struktur GVSTreeWithCollection benutzt.	Leerer Screen wird dargestellt. Name der Session erscheint im Dropdown Menü.	+	Braucht es hier eine UserInfo à la "empty Graph"?
5	CyclicRootCollection.java	Darstellung eines Trees mittels Collection. Die Knoten haben Zyklen untereinader.	Die Zyklen werden Clientseitig erkannt und die Übertragung abgebrochen. Der Server wird freigegeben.	+	

Systemtest: GVS UI Allgemein

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.*	-	Nicht Bestanden
Datum	04.12.17		

Testergebnis ✔ 100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	-	Löschen einer Session	Die Session wird nicht mehr angezeigt und ist im Dropdown Menü nicht mehr aufgelistet. Anzeige wechselt zur aktuellsten Session.	+	
2	-	Löschen der letzten Session	Das Fenster wechselt zur Startansicht. Sämtliche Session spezifischen Controls werden ausgeblendet und das Background-Logo wird eingeblendet. Die Speicher und Delete Buttons sind deaktiviert.	+	
3	relativeGraph.gvs	Laden einer bereits geladenen Session	Die Session wird nicht ein zweites Mal geladen. Ist gerade eine andere Session aktiv, wird zur bestehenden Session gewechselt, welche der geladenen Session entspricht.	+	
4	emptySessionName .java	Darstellen einer Session mit leerem Namen	Session wird mit leeren String ins Dropdown aufgenommen. Alles funktioniert wie gewohnt.	+	
5	-	Wechseln der aktuellen Session über die Dropdown Liste	Die ausgewählte Session wird angezeigt	+	

Systemtest: Client - Server Connection

System under Test	https://github.com/Graphs-Visualization-Service
Package	gvs.tester.*
Datum	04.12.17

\+ Bestanden
\- Nicht Bestanden

Testergebnis ✔ 100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	relativeGraph.java	Versuchter Verbindungsaufbau eines zweiten Clients	Die Verbindung mit dem ersten Client wird nicht gestört. Der zweite Client kommt in die Warteschlaufe.	+	
2	watchdogTest.java	Verbindungsabbruch des Clients (z.B. NullPointer)	Der reservierte Server wird wieder frei gegeben.	+	

Systemtest: Graph - Normale Fälle

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.graph	-	Nicht Bestanden
Datum	15.12.17		

Testergebnis ✔106%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	RelativeGraph.java	Darstellung eines Graphen mit relativen Vertices, deren Positionen vom User fixiert sind.	Koordinaten der Vertices werden prozentual interpretiert.	+	
2	DefaultGraph.java	Darstellung eines Graphen mit default Vertices, deren Positionen vom User nicht angegeben werden.	Koordinaten der Vertices werden vom Layouter für den ersten Graphen berechnet. Für weitere Graphen der gleichen Session werden die Koordinaten übernommen.	+	
3	RelativeGraph.java DefaultGraph.java	Darstellung von gerichteten Edges.	Edge wird als Pfeil dargestellt. Der Schnittpunkt des Pfeils mit dem Vertex ist korrekt berechnet und verändert sich beim Dragging des Vertex entsprechend.	+	Es wäre allenfalls etwas schöner (v.a. bei kurzen Edges), wenn das Label mittig zur linie "ohne Pfei" wäre
4	RelativeGraph.java DefaultGraph.java	Darstellung von ungerichteten Edges ohne Pfeilangabe.	Edge wird als Linie dargestellt. Der Schnittpunkt der Linie mit dem Vertex ist korrekt berechnet und verändert sich beim Dragging des Vertex entsprechend.	+	
5	RelativeGraph.java DefaultGraph.java	Darstellung von ungerichteten Edges mit Pfeilangabe.	Edge wird als Pfeil dargestellt. Der Schnittpunkt des Pfeils mit dem Vertex ist korrekt berechnet und verändert sich beim Dragging des Vertex entsprechend.	+	
6	IconGraph.java	Darstellung eines Graphen mit Default Vertices, welche teilweise ein Label mit Icon und teilweise nur ein Icon enthalten.	Korrekte Darstellung des Graphen.	+	
7	IconGraphRelative .java	Darstellung eines Graphen mit Relative Vertices, welche teilweise ein Label mit Icon und teilweise nur ein Icon enthalten.	Korrekte Darstellung des Graphen.	+	
8	RelativeGraph.java DefaultGraph.java	Darstellung einer Abfolge von Graphen mit Styleänderungen.	Korrekte Darstellung der Styles.	+	
9	RelativeGraph.java DefaultGraph.java	Darstellung einer Abfolge von Graphen mit Hinzufügen und Entfernen von Vertices.	Korrekte Darstellung des Graphen. Bestehende Vertices übernehmen die Positionen des vorherigen Graphen.	+	
10	RelativeGraph.java DefaultGraph.java	Abspeicherung einer Graph Session.	Die Session wird korrekt gespeichert. Der Speicherort wird vom User gewählt. Die korrekte Dateiendung (*.gvs) wird gesetzt. Die berechneten Koordinaten werden	+	

Systemtest: Graph - Spezielle Fälle

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.graph	-	Nicht Bestanden
Datum	15.12.17		

Testergebnis ✔ 100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	LongLabels.java	Darstellung eines Graphen mit langen Vertex Labels.	Die langen Labels werden abgekürzt. Beispiel: LongLabel -> Lo....el	+	
2	mixedGraph.java	Darstellung eines Graphen der Default Vertices und relative Vertices enthält.	Client wirft passende Exception und sendet keine Daten an den Server	+	
3	emptyGraph.java	Übermittlung eines Graphen der keine Vertices enthält.	Leerer Screen wird dargestellt. Name der Session erscheint im Dropdown Menü.	+	
4	nullEdges.java	Übermittlung eines Graphen der Edges enthält, welche null als Vertex-Start oder Ende besitzen.	Edges sollen nicht angezeigt werden. Es kommt zu keinen Fehlern im GVS 2.0 UI	+	
5	emptyString.java	Darstelllung eines Graphen mit Vertices, welche leere Strings als Labels besitzen.	Vertices werden trotzdem angezeigt. Sie besitzen kein Label.	+	

Systemtest: Tree - Normale Fälle

System under Test	https://github.com/Graphs-Visualization-Service
Package	gvs.tester.tree
Datum	15.12.17

\+ Bestanden
\- Nicht Bestanden

Testergebnis ✔ 100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	BinaryTree.java	Dartellung eines Baums mit binären Nodes. Dabei sollen gewisse Nodes nur ein linkes oder nur ein rechtes Kind besitzen.	Platzierung der Nodes wird vom Layouter richtig berechnet. AutoLayout sowie Dragging ist deaktiviert.	+	
2	MultipleRoots.java	Darstellung eines Forests	Die einzelnen Bäume überlappen sich nicht.	+	
3	BinaryTree.java	Darstellung einer Abfolge von Trees mit Hinzufügen und Löschen von Nodes.	Korrekte Darstellung der Session	+	
4	BinaryTree.java	Darstellung einer Abfolge von Trees mit Styleänderungen.	Korrekte Darstellung der Styles. Edges übernehmen immer den Style des Kind Nodes.	+	
5	DefaultTree.java	Darstellung eines n-ary Trees mit mehreren Levels und unterschiedlichen Anzahlen von Kindern	Korrekte Darstellung der Session	+	
6	BinaryTree.java	Abspeicherung einer Tree Session.	Die Session wird korrekt gespeichert. Der Speicherort wird vom User gewählt. Die korrekte Dateiendung (*.gvs) wird gesetzt.	+	
7	test.gvs	Laden einer Tree Session	Die Session wird korrekt geladen. Es werden nur Dateien mit der *.gvs Endung angezeigt.	+	
8	BinaryTree.java	Replay einer Tree Session	Die Session wird korrekt abgespielt. Während dem Abspielen wird der Play Button zum Pause Button. Nur der Pause und der Cancel Button sind während dem Abspielen aktiv. Graphen können von jedem Step aus abgespielt werden. Der Step wird beim Pausieren beibehalten. Beim Cancel springt die Session auf den ersten Graphen.	+	
9	BinaryTree.java	Durchsteppen eines Trees	Die Step Buttons (First, Prev, Next, Last) navigieren korrekt durch die Session. Bei den ersten und letzten Graphen sind die korrekten Buttons aktiviert/deaktiviert. Die Progressbar zeigt den korrekten Füllstand.	+	
10	BinaryTree.java	DragSupport eines Trees	DragSupport für Trees ist deaktiviert.	+	
11	BinaryTree.java	AutoLayout eines Trees	AutoLayout für Trees ist deaktiviert.	+	
12	StyleTester.java	Sämtliche unterstützten Styles für Edges und Nodes werden übertragen und dargestellt (Farben, Liniendicke, Linienstyle)	Alle Styles werden wie im Client definiert dargestellt	+	
13	ClusterSplitter	Darstellung überfüllter Trees	Baum wird trotz vieler Nodes schön dargestellt.	+	

Systemtest: Tree - Spezielle Fälle

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.tree	-	Nicht Bestanden
Datum	15.12.17		

Testergebnis ✔ 100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	LongLabels.java	Darstellung eines Trees mit langen Vertex Labels.	Die langen Labels werden abgekürzt. Beispiel: LongLabel -> Lo....el	+	
2	EmptyString.java	Darstelllung eines Trees mit Vertices, welche leere Strings als Labels besitzen.	Nodes werden trotzdem angezeigt. Sie besitzen kein Label.	+	
3	RemoveRoot.java	Darstellung einer Tree Abfolge, wenn die Root entfernt wird. Dabei wird in der GVS 2.0 Lib die Struktur GVSTreeWithRoot benutzt.	Leerer Screen wird dargestellt. Name der Session erscheint im Dropdown Menü.	+	
4	RemoveRootCollection.java	Darstellung einer Tree Abfolge, wenn die Root entfernt wird. Dabei wird in der GVS 2.0 Lib die Struktur GVSTreeWithCollection benutzt.	Leerer Screen wird dargestellt. Name der Session erscheint im Dropdown Menü.	+	
5	CyclicRootCollection.java	Darstellung eines Trees mittels Collection. Die Knoten haben Zyklen untereinader.	Die Zyklen werden Clientseitig erkannt und die Übertragung abgebrochen. Der Server wird freigegeben.	+	

Systemtest: GVS UI Allgemein

System under Test	https://github.com/Graphs-Visualization-Service
Package	gvs.tester.*
Datum	15.12.17

\+ Bestanden
\- Nicht Bestanden

Testergebnis ✔ 100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	-	Löschen einer Session	Die Session wird nicht mehr angezeigt und ist im Dropdown Menü nicht mehr aufgelistet. Anzeige wechselt zur aktuellsten Session.	+	
2	-	Löschen der letzten Session	Das Fenster wechselt zur Startansicht. Sämtliche Session spezifischen Controls werden ausgeblendet und das Background-Logo wird eingeblendet. Die Speicher und Delete Buttons sind deaktiviert.	+	
3	relativeGraph.gvs	Laden einer bereits geladenen Session	Die Session wird nicht ein zweites Mal geladen. Ist gerade eine andere Session aktiv, wird zur bestehenden Session gewechselt, welche der geladenen Session entspricht.	+	
4	emptySessionName.java	Darstellen einer Session mit leerem Namen	Session wird mit leeren String ins Dropdown aufgenommen. Alles funktioniert wie	+	
5	-	Wechseln der aktuellen Session über die Dropdown Liste	Die ausgewählte Session wird angezeigt	+	

Systemtest: Client - Server Connection

System under Test	https://github.com/Graphs-Visualization-Service	+	Bestanden
Package	gvs.tester.*	-	Nicht Bestanden
Datum	15.12.17		

Testergebnis ✔ 100%

Test-Nr.	Test File	Test Beschreibung	Erwartetes Resultat	Ergebnis	Bemerkungen
1	relativeGraph.java	Versuchter Verbindungsaufbau eines zweiten Clients	Die Verbindung mit dem ersten Client wird nicht gestört. Der zweite Client kommt in die Warteschlaufe.	+	
2	watchdogTest.java	Verbindungsabbruch des Clients (z.B. NullPointer)	Der reservierte Server wird wieder frei gegeben.	+	

E.2 Usability Tests

Viele Funktionen sind im GVS 1.0 versteckt und es ist nicht immer klar, was eine Funktion auslöst. Beim Design von GVS 2.0 wurde deshalb darauf geachtet, dass ein minimales, sprechendes UI erstellt wird. Beim Vergleich der beiden Programmfenster (siehe Abb. E.1 und E.2) sind die Unterschiede gut erkennbar.

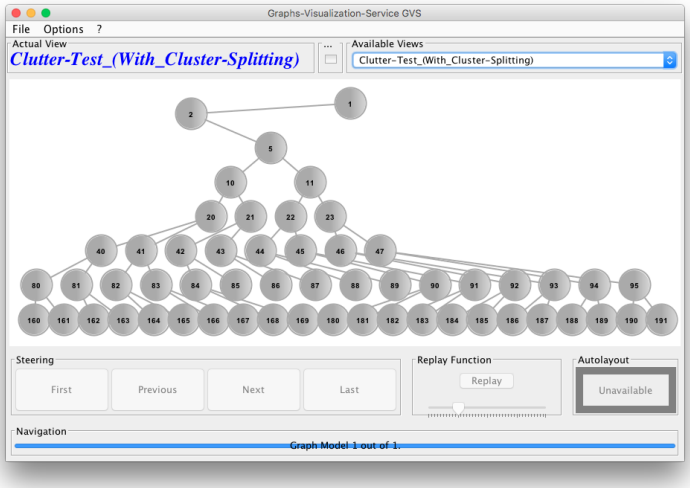


Abbildung E.1 – GVS 1.0: User Interface

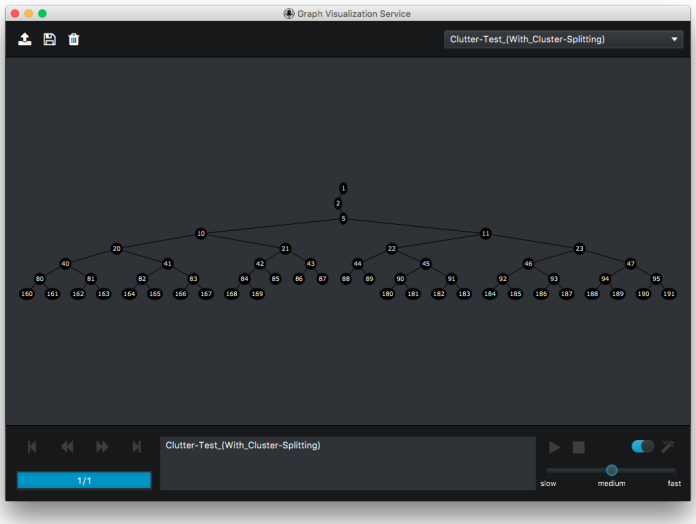


Abbildung E.2 – GVS 2.0: User Interface

Erkannte Mängel GVS 1.0	Einfluss auf GVS 2.0
Veraltetes User Interace	Modernes, ansprechendes User Interface, mit dunklen Farben, damit die Algorithmen besser farblich hervorgehoben werden
Icons werden unter Unix nicht dargestellt	Moderne, sprechende FontAwesome Icons eingeführt. Die Icons werden auf allen Plattformen korrekt dargestellt
Funktion *Random und Stable Layout* ist nicht klar	Die Funktion wird wegen fehlendem praktischen Nutzen entfernt
Funktion *Cluster Splitter* ist nicht klar	Ein spezieller Layout Algorithmus für Binary Trees wird implementiert
Beim Laden von gespeicherten Sessions muss der Speicherort immer wieder von Neuem ausgewählt werden	Der zuletzt verwendete Speicherort wird gespeichert
Graph Vertices lassen sich ausserhalb des sichtbaren Programmfenster draggen	ZoomPane skaliert das Fenster, sobald ein Knoten den Viewport verlässt
Autolayout Funktion ist nur für die letzten Momentaufnahme eines Graphen möglich	Die Autolayout Funktion ist für alle Momentaufnahmen möglich
Drag Support ist nur für die letzte Momentaufnahme eines Graphen möglich	Dragging von Graph Vertices ist immer möglich
Koordinaten von gedraggten Vertices werden nicht für die nächste Momentaufnahme übernommen	Koordinaten werden immer übernommen
Replay Geschwindigkeit wird in Millisekunden dargestellt	Es gibt nur doch die drei Geschwindigkeiten *langsam*, *normal* und *schnell*
Server Applikation wird beim Absturz des Client nicht freigegeben	Watchdog Service gibt den Server bei inaktiven Clients wieder frei

Tabelle E.1 – Usability Tests

Anhang F

Zeitauswertung

F.1 Auswertung nach Teammitgliedern

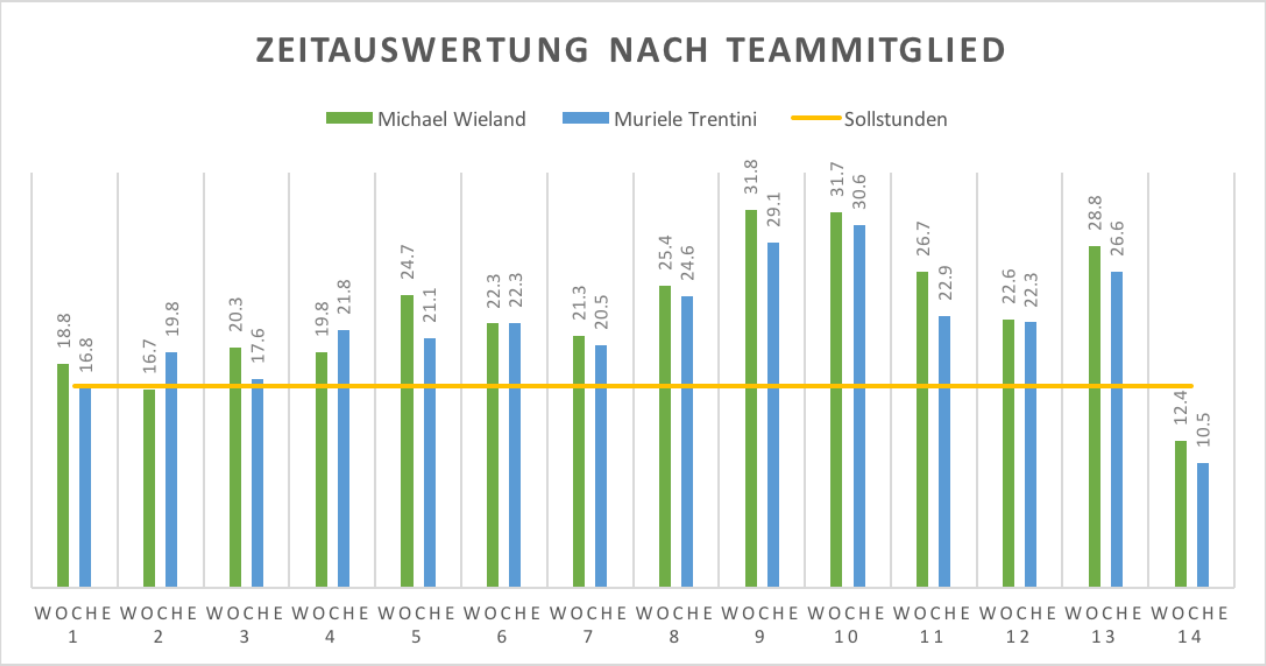


Abbildung F.1 – Zeitauswertung Teammitglieder

F.2 Auswertung nach Kategorien

Administratives

Meetings mit dem Betreuer, Sprint-Meetings, Verfassung von Protokollen, Aufsetzen der Entwicklungsumgebung & Projektmanagement Software, Software Releases

Dokumentation

Verfassen der Dokumentation, Planen der Architektur, Durchführen von End-to-End Tests

Implementierung

Schreiben von Code, inkl. Refactorings & Unit Tests

Requirement Analyse

Analyse des bestehenden Legacy Codes

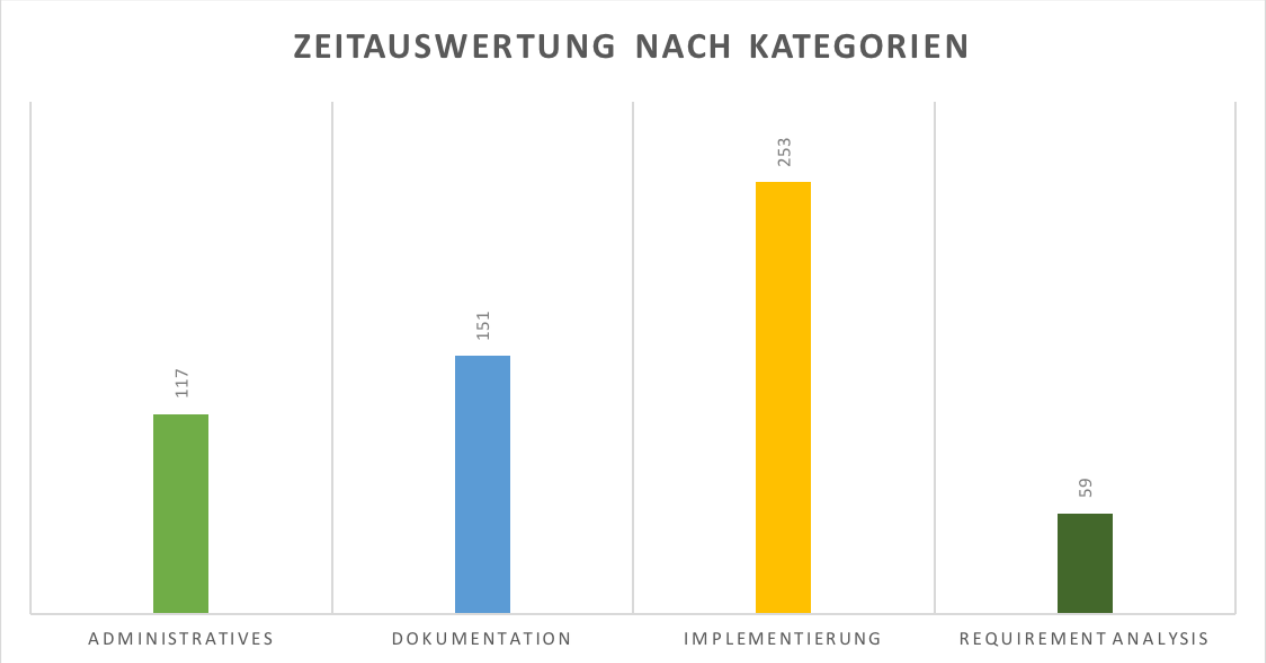


Abbildung F.2 – Zeitauswertung Kategorien Säulendiagramm

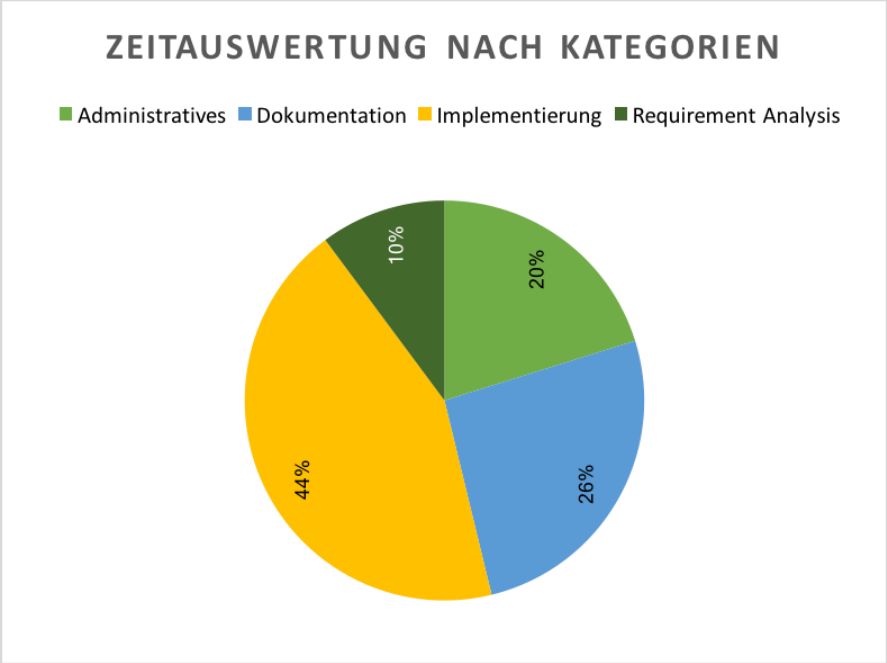


Abbildung F.3 – Zeitauswertung Kategorien Kuchendiagramm

F.3 Auswertung Soll - Ist Zeit

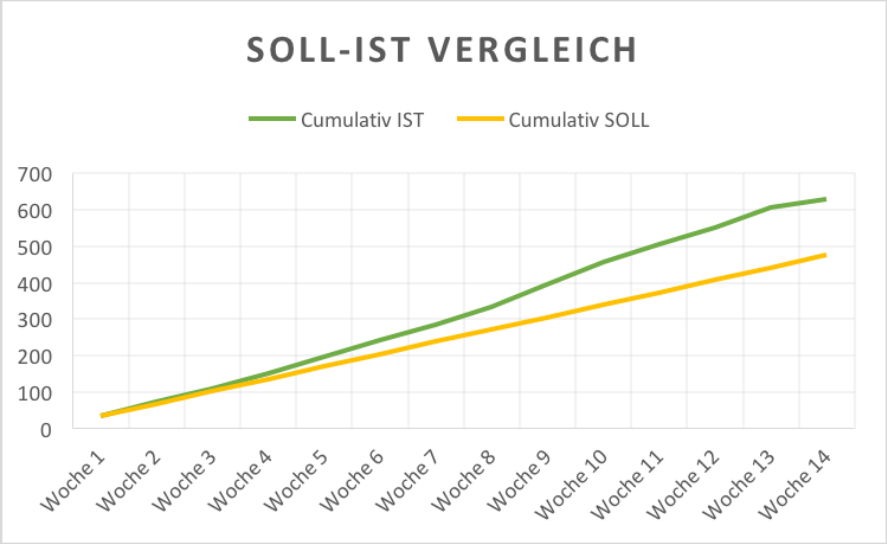
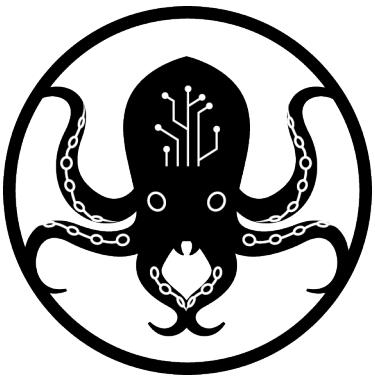


Abbildung F.4 – Zeitauswertung Soll - Ist Zeit

Anhang G

Benutzerhandbuch

Nachfolgend findet sich das Benutzerhandbuch für GVS 2.0 inklusive einer Installationsanleitung für die Benutzer der Library. Für die Weiterentwicklung werden die nötigen Schritte für das Aufsetzen der Entwicklungsumgebung beschrieben.



Graph-Visualization-Service GVS 2.0

Benutzerhandbuch

Autoren
Michael Wieland
Murièle Trentini

Inhaltsverzeichnis

Kapitel 1

Anforderungen und Hinweise

Für die Inbetriebnahme von GVS 2.0 sind folgende Hinweise zu beachten.

- GVS 2.0 benötigt JDK 1.8 oder neuer
- Für die Entwicklung wurde die Eclipse IDE for Java Developers - Oxygen und Visual Studio-IDE 2017 verwendet
- Für die Weiterentwicklung des GVS 2.0 werden Grundkenntnisse von git vorausgesetzt
- Für die Weiterentwicklung des GVS 2.0 werden Grundkenntnisse für die Plugin Installation in Eclipse vorausgesetzt
- Die folgende Installationsanleitung wurde für die Eclipse IDE geschrieben. Die Weiterentwicklung sollte aber grundsätzlich auch in anderen IDE's funktionieren.

Kapitel 2

Installationsanleitung

2.1 GVS Lib im Projekt integrieren

Um GVS 2.0 nutzen zu können, muss die GVS Lib als Dependency in deinem Java Projekt hinzugefügt werden.

1. Rechtsklick auf das Projekt im *Package Explorer*
2. `Build Path > Configure Build Path...`
3. `Tab ''Libraries''> Add External JAR...`
4. *gvs-lib-java.jar* in Ordnerstruktur auswählen
5. mit `Apply and Close` bestätigen

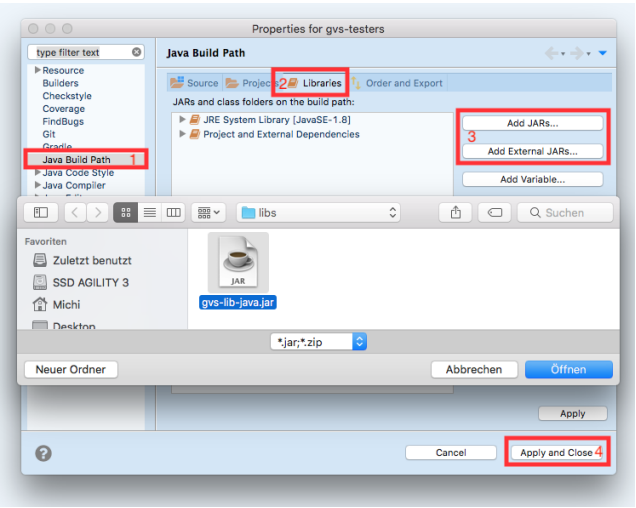


Abbildung 2.1 – GVS Lib im Class Path verfügbar machen

2.2 GVS UI ausführen

2.2.1 Windows & macOS

Das JAR File kann entweder direkt per Doppelklick oder über die Konsole ausgeführt werden.

```
java -jar gvs-ui.jar
```

2.2.2 Linux

Oracle JDK Für Linux Distributionen, die *oraclejdk* verwenden, kann das JAR File ebenfalls direkt über die Konsole ausgeführt werden.

```
java -jar gvs-ui.jar
```

Open JDK Für Linux Distributionen, die *openjdk* verwenden, wird zusätzlich die Installation von *openjfx* benötigt.

```
sudo dnf install java-1.8.0-openjdk-openjfx
java -jar gvs-ui.jar
```

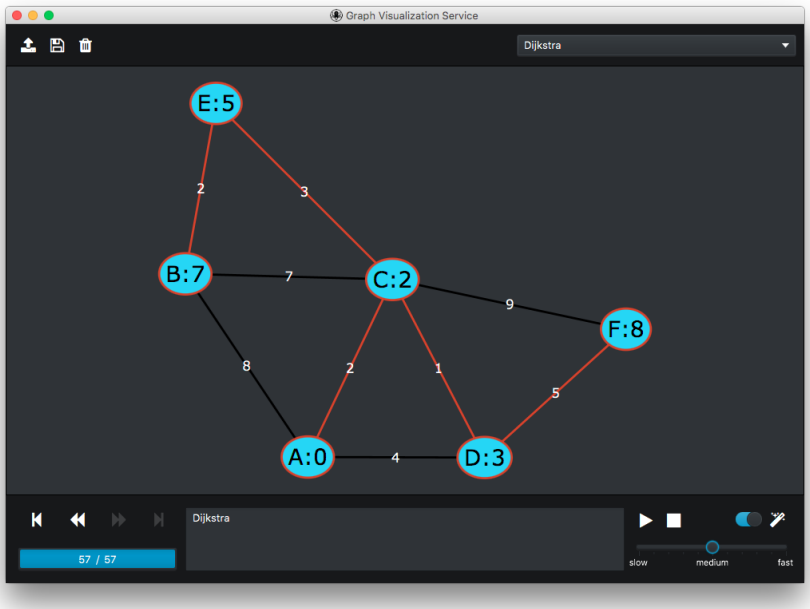


Abbildung 2.2 – GVS UI

Kapitel 3

Benutzeranleitung

3.1 GVS Lib

Die folgenden Testfiles sowie Referenzimplementierungen sind im Repository gvs-tester zu finden (siehe 4.1).

3.1.1 Graphen

Zur Modellierung von Graph Elementen stehen folgende Klassen zur Verfügung.

Interface	Nutzen
GVSGraph	Ein Graph
GVSDefaultVertex	Ein Vertex ohne Koordinaten.
GVSRelativeVertex	Ein Vertex mit fixen Koordinaten (in Prozent)
GVSDirectedEdge	Eine gerichtete Kante für Graphen
GVSUndirectedEdge	Eine ungerichtete Kante für Graphen

Tabelle 3.1 – GVS Lib 2.0 Graph Interfaces

Der Code Ausschnitt 3.1 zeigt ein Beispiel, wie ein Graph erstellt und an das GVS UI gesendet werden kann.

Listing 3.1 – Hello World Graph

```
public class HelloGraph {

    public static void main(String[] args) {
        GVSGraph graph = new GVSGraph("Hello World Graph");
        TestDefaultVertex v1 = new TestDefaultVertex("V1");
        TestDefaultVertex v2 = new TestDefaultVertex("V2");
        TestDirectedEdge e = new TestDirectedEdge(v1, v2, "V1 to V2");

        graph.add(v1);
        graph.add(v2);
        graph.add(e);

        graph.display(); // send to gvs ui
        graph.disconnect();
    }
}
```

Durch jeden Aufruf von `graph.display()` wird ein Snapshot des Graphen an das GVS UI gesendet. Ganz zum Schluss des Programms muss `graph.disconnect()` aufgerufen werden.

Vertices und Edges

Für Vertices und Edges müssen ebenfalls konkrete Instanzen der Interfaces erstellt werden. Der Code Auszug 3.2 zeigt ein Beispiel einer konkreten *GVSDefaultVertex* Implementierung.

Listing 3.2 – DefaultVertex Implementierung

```
public class TestDefaultVertex implements GVSDefaultVertex {
    private String label;
    private GVSStyle style;

    public TestDefaultVertex(String label) {
        this.label = label;
    }

    @Override
    public String getGVSVertexLabel() {
        return label;
    }

    @Override
    public GVSStyle getStyle() {
        return style;
    }

    public void setStyle(GVSStyle style) {
        this.style = style;
    }
}
```

3.1.2 Trees

Zur Modellierung von Tree Elementen stehen folgende Klassen zur Verfügung.

Interface	Nutzen
GVSTreeWithRoot	Erstellt einen Tree anhand der Root Node und deren Kind Beziehungen
GVSTreeWithCollection	Ein Forest bestehend aus mehreren Trees.
GVSBinaryNode	Ein Vertex für Binary Trees

Tabelle 3.2 – GVS Lib 2.0 Tree Interfaces

Der Code Ausschnitt 3.3 zeigt ein Beispiel, wie ein Tree erstellt und an das GVS UI gesendet werden kann.

Listing 3.3 – Hello World Tree

```
public class HelloTree {

    public static void main(String[] args) {
        GVSTreeWithRoot tree = new GVSTreeWithRoot("Hello World Tree");

        TestBinaryNode root = new TestBinaryNode("root");
        TestBinaryNode left = new TestBinaryNode("left");
        TestBinaryNode right = new TestBinaryNode("right");

        root.setLeftChild(left);
        root.setRightChild(right);

        tree.setRoot(root);

        tree.display();
        tree.disconnect();
    }
}
```

Binary Nodes

Der Code Auszug 3.4 zeigt ein Beispiel einer konkreten *GVSBinaryTreeNode* Implementierung. Die Edges werden vom UI implizit gezeichnet.

Listing 3.4 – Binary Node Implementierung

```
public class TestBinaryNode implements GVSBinaryTreeNode {
    String label;
    GVSStyle style;
    GVSBinaryTreeNode leftChild;
    GVSBinaryTreeNode rightChild;

    public TestBinaryNode(String label, GVSStyle style) {
        this.label = label;
        this.style = style;
    }

    public TestBinaryNode(String label) {
        this(label, null);
    }

    @Override
    public String getNodeLabel() {
        return label;
    }

    @Override
    public GVSStyle getStyle() {
        return style;
    }

    @Override
    public GVSBinaryTreeNode getGVSLeftChild() {
        return leftChild;
    }

    @Override
    public GVSBinaryTreeNode getGVSRightChild() {
        return rightChild;
    }

    public void setRightChild(GVSBinaryTreeNode child) {
        this.rightChild = child;
    }

    public void setLeftChild(GVSBinaryTreeNode child) {
        this.leftChild = child;
    }

    public void setStyle(GVSStyle style) {
        this.style = style;
    }
}
```

3.1.3 Styles anpassen

Um komplexe Algorithmen farblich zu visualisieren bietet GVS 2.0 eine Style Klasse an. Code Ausschnitt 3.5 zeigt die Anpassungsmöglichkeiten von Vertices und Edges.

Listing 3.5 – Styles verändern

```
// lineColor, lineStyle, lineThickness
GVSStyle edgeStyle = new GVSStyle(GVSColor.RED, GVSLineStyle.DASHED,
GVSLineThickness.BOLD);

// lineColor, lineStyle, lineThickness, fillColor
GVSStyle vertexStyle = new GVSStyle(GVSColor.BLUE, GVSLineStyle.DOTTED,
GVSLineThickness.SLIGHT, GVSColor.GREEN);

v1.setStyle(vertexStyle);
e.setStyle(edgeStyle);
```

3.1.4 Icons verwenden

GVS 2.0 nutzt zur Darstellung von Vertex Icons die Schriftart FontAwesome. Code Abschnitt 3.6 zeigt, wie die Icons für Vertices verwendet werden können.

Listing 3.6 – Icons benutzen

```
// lineColor, lineStyle, lineThickness, fillColor, icon
GVSStyle iconStyle = new GVSStyle(GVSColor.STANDARD, GVSLineStyle.THROUGH,
GVSLineThickness.STANDARD, GVSColor.STANDARD, GVSIcon.BICYCLE);
v1.setStyle(iconStyle);
graph.display();
```

3.2 GVS UI Benutzeroberfläche

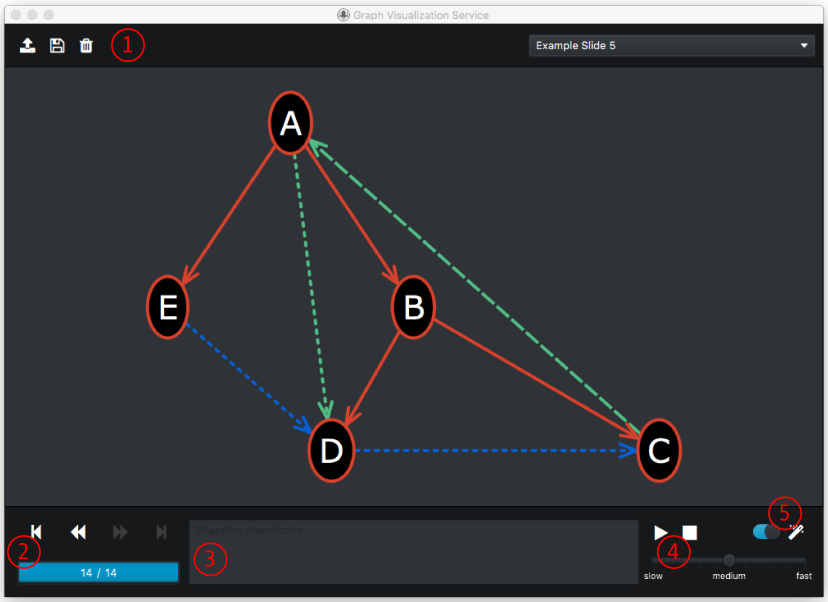


Abbildung 3.1 – User Interface GVS 2.0

3.2.1 User Interface

Alle wichtigen Funktionen von GVS 2.0 sind direkt über die Toolbar zugreifbar. Alle Buttons verfügen über Tooltips.

1. Toolbar

Über die Toolbar, die am oberen Fensterrand dargestellt wird, sind die grundlegenden Aktionen schnell und einfach zugänglich. Vier Aktionen sind möglich:

1. Session laden: GVS Session im `*.gvs` Format vom Dateisystem laden
2. Session speichern: Aktuelle Session auf das Dateisystem speichern
3. Session löschen: Aktuelle Session löschen. Diese Aktion hat keinen Effekt auf das Dateisystem.
4. Session wechseln: In der Dropdown Liste oben rechts, werden alle aktiven Sessions dargestellt. Zwischen den Sessions kann beliebig gewechselt werden.

2. Step Buttons und Fortschrittsanzeige

Die Step Buttons bilden zusammen mit der Replay Funktionalität ein wichtiges Werkzeug, um die Funktionsweise eines Algorithmus schrittweise zu analysieren und zu verstehen. Über die Buttons können die einzelnen Momentaufnahmen einer Session durchgeschaut werden.

3. Snapshot Description

Zur aktuellen Momentaufnahme können Notizen erfasst werden. Beim Speichern der Session werden die Notizen ebenfalls gespeichert und beim Laden der Session wieder angezeigt.

4. Replay

Die Replay Funktionalität automatisiert die Aktionen der Step Buttons in einer bestimmten Geschwindigkeit. Standardmässig wird jede Sekunde zur nächsten Momentaufnahme gewechselt. Das Replay kann nach belieben gestoppt, gestartet und abgebrochen werden.

5. Autolayout

Das Autolayout steht nur für Graphen zur Verfügung. Vertices werden so positioniert, dass es möglichst wenig Überschneidungen der Edges gibt. Über den Umschaltknopf kann bestimmt werden, ob das Layout forciert wird. Ist der Toggle Button deaktiviert, werden nur noch jene Vertices gelayouted, die noch nicht mit der Maus manuell positioniert wurden.

3.2.2 Drag Support

Der Drag Support steht nur für Graphen zur Verfügung. Vertices können mit der Maus beliebig positioniert werden. Wenn ein Vertex aus dem aktuellen ViewPort gezogen wird, zoomt das Fenster automatisch heraus.

3.3 Konfiguratiosfiles verändern

Um Änderungen an Konfigurationsfiles durchzuführen, die im JAR File enthalten sind, müssen folgende Schritte befolgt werden.

1. JAR File mit einem beliebigen File Archiver entpacken
2. In den entstandenen Ordner wechseln
   (a) `cd gvs-ui`
3. gewünschte Files verändern
4. Files wieder packetieren
   (a) `jar cfm ../gvs-ui.jar META-INF/MANIFEST.MF gvs/GVSApplication.class *`

Folgende Einstellungen können vorgenommen werden:

File	Einstellung
config.xml	Festlegen des Server Socket Startports. Default: `3000`
gvs/ui/view/app/AppView.css	Farben, Dicke von Edges und Vertex-Ränder
gvs/ui/view/session/SessionView.css	Schriftgrösse für Vertices und Edges
src/main/resources/logback.xml	Festlegen des Log-Levels (siehe 3.3.1)

Tabelle 3.3 – GVS Lib 2.0 Interfaces

3.3.1 Ändern des Log-Level

Im XML File

Die XML Konfigurationsdatei befindet sich unter `src/main/resources/logback.xml`.
Das applikations- übergreifende Log Level kann über das Root Element bestimmt werden:

Listing 3.7 – Root Log Level verändern

```
<root level="DEBUG"> <!-- Log Level  -->
    <appender-ref ref="FILE" /> <!--  Log to File -->
    <appender-ref ref="STDOUT" /> <!--  Log to Standard Output-->
</root>
```

Es ist auch möglich, dass Log Level für einzelnen Packages zu setzen. Dazu muss ein neues Logger Element erstellt werden.

Listing 3.8 – Package Log Level verändern

```
<logger name="gvs.access" level="DEBUG">
    <appender-ref ref="STDOUT" /> <!--  Log to Standard Output-->
</logger>
```

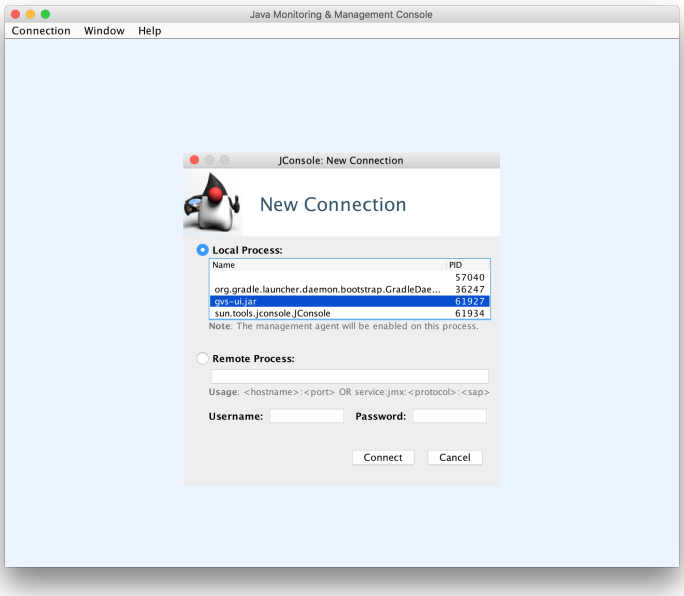
Mögliche Level sind:

- OFF
- TRACE
- DEBUG
- INFO
- WARN
- ERROR

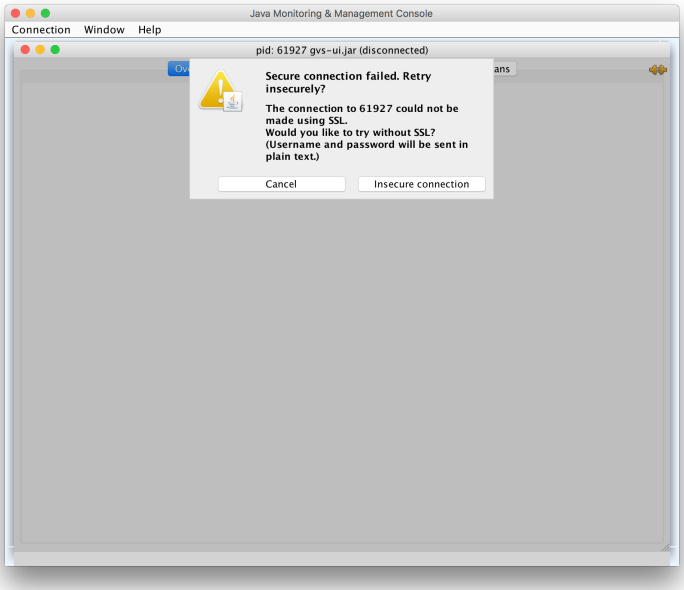
Zur Laufzeit

Über die JConsole kann das Log Level auch zur Laufzeit verändert werden. Die JConsole kann einfach über die Konsole geöffnet werden:

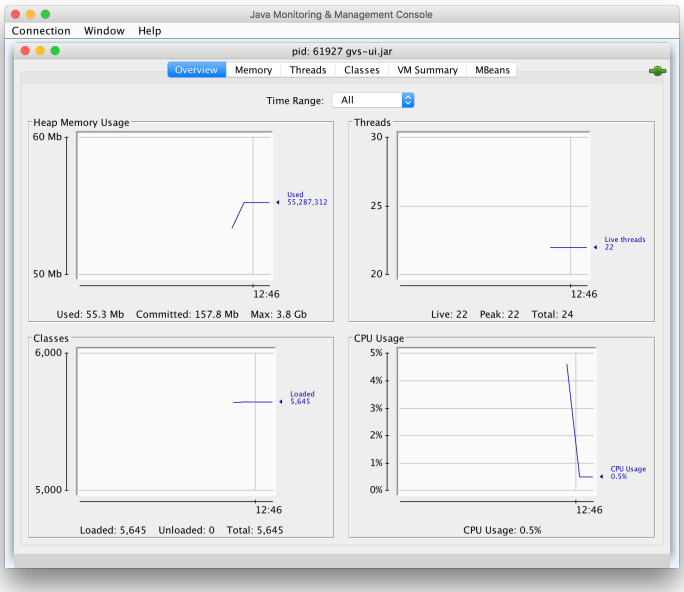
```
$bash> jconsole
```



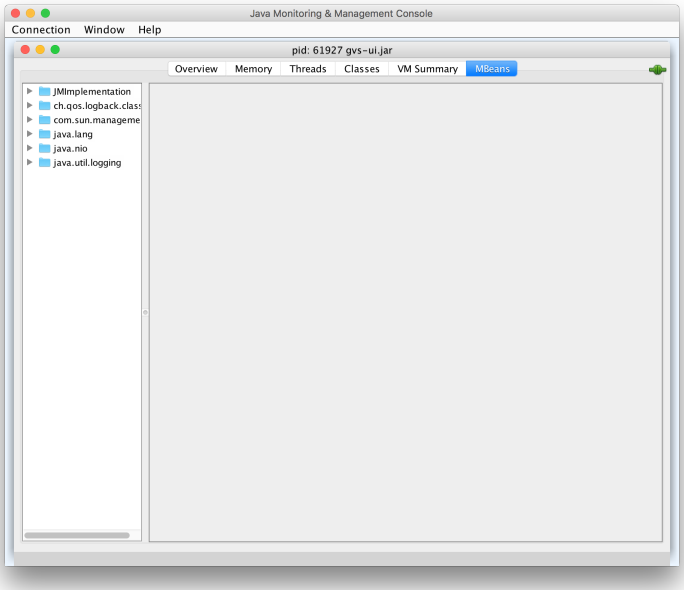
gvs-ui aus Liste auswählen und mit `Connect` bestätigen



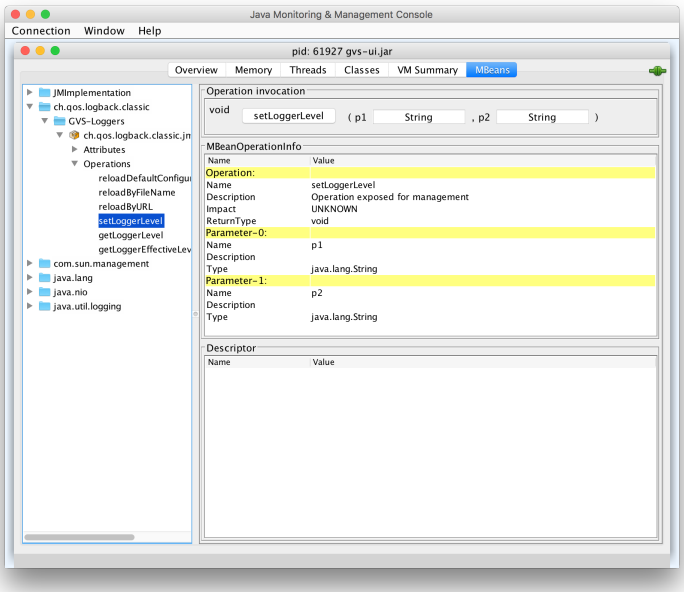
Warnung über unsichere Verbindung bestätigen



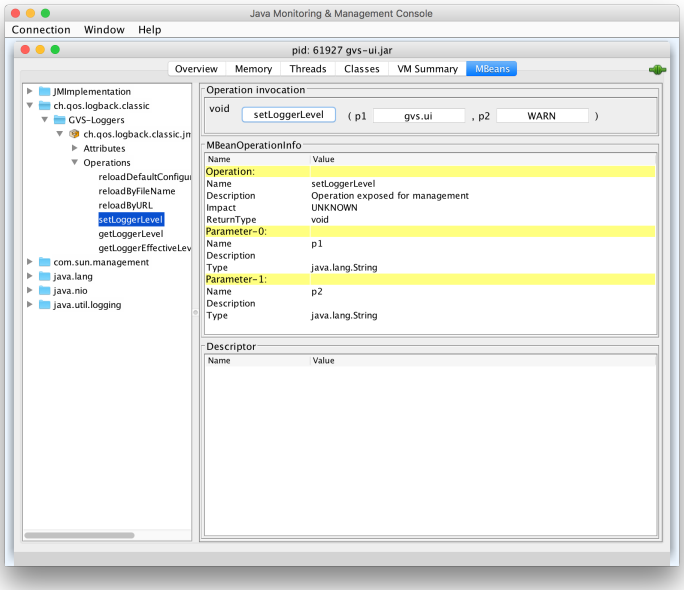
Ansicht *Overview* erscheint



Zu Ansicht MBeans wechseln



Links im Verzeichnis-Baum zu `ch.qos.logback.classic` > `GVS-Loggers` > `Operations` > `setLoggerLevel` wechseln



Im oberen Fenster-Bereich unter *Operation invocation* gewünschte Parameter angeben

1. Parameter: package (e.g. `gvs.ui`)
2. Parameter: Log Level (e.g. `WARN`)

Mit `setLoggerLevel` bestätigen

Kapitel 4

Weiterentwicklung

Sämtlicher Programmcode von GVS 2.0 ist frei auf Github verfügbar. Um Verbesserungen durchzuführen beschreibt dieser Abschnitt die notwendigen Schritte, um sich die Entwicklungsumgebung entsprechend einzurichten.

4.1 Repositories

GVS 2.0 umfasst 4 Repositories, welche in Tabelle 4.1 aufgeführt sind.

Repository	Nutzen	URL
gvs-ui	UI und Socket Server	https://github.com/Graphs-Visualization-Service/gvs-ui
gvs-lib-java	Library für Java Programme	https://github.com/Graphs-Visualization-Service/gvs-lib-java
gvs-lib-csharp	Library für C# Programme	https://github.com/Graphs-Visualization-Service/gvs-lib-csharp
gvs-tester	Ausführbare End-to-End Testfiles in Java	https://github.com/Graphs-Visualization-Service/gvs-tester

Tabelle 4.1 – GVS 2.0 Repositories

4.2 Entwicklungsumgebung

4.2.1 Java

Für die Entwicklung des GVS 2.0 UI und der GVS 2.0 Java Lib wurde die *Eclipse IDE for Java Developers* in der Version *Oxygen* verwendet. Zusätzlich wurden folgende Plugins installiert.

Eclipse Plugins

Sämtliche Plugins können optional installiert werden. Es empfiehlt sich aber mindestens Gradle in Eclipse zu integrieren.

Buildship Gradle Integration in Eclipse [1]

EGit Git Integration in Eclipse [2]

Checkstyle Codestyle Überprüfung in Eclipse [3]

FindBugs Statische Codeanalyse in Eclipse [4]

Stan4J Strukturanalyse in Eclipse [5]

ECLEmma Test Coverage Analyse in Eclipse [6]

Gluon Scene Builder

Für die Entwicklung von JavaFX Interfaces kann der Scene Builder von Gluon [7] verwendet werden. Wie die meisten WYSIWYG Editoren ist auch der Scene Builder sehr verbose beim Erstellen von XML Elementen. Es empfiehlt sich, eine grundlegende Struktur im Scene Builder zu erstellen und diese dann im XML Editor von Eclipse anzupassen.

4.2.2 C Sharp

Für die GVS 2.0 C# Lib wurden die Visual Studio-IDE 2017 [8] ohne weitere Plugins verwendet.

1 https://projects.eclipse.org/projects/tools.buildship
2 https://eclipse.org/egit
3 http://eclipse-cs.sf.net
4 http://findbugs.sourceforge.net
5 http://stan4j.com
6 http://www.eclemma.org
7 http://gluonhq.com/products/scene-builder
8 https://www.visualstudio.com

4.3 Schritt für Schritt Integration

Am Beispiel des gvs-ui Repositories folgt eine Schritt für Schritt Anleitung vom git clone bis zum fertigen Build. Diese Anleitung kann analog auch für die anderen Java Repositories benutzt werden.

1. `git clone https://github.com/Graphs-Visualization-Service/gvs-ui.git`
2. Importieren des Projekts in Eclipse
   (a) `File > Import > Existing Gradle Project`
   (b) als *Project root directory* das geklonte Repository auswählen und mit `Finish` bestätigen.
3. Nötige Dependencies installieren
   (a) Rechtsklick auf den Projekt-Ordner im *Package Explorer*
   (b) `Gradle > Refresh Gradle Project`
4. Builden des Projekts mit Gradle
   (a) `Window > Show View > Other`
   (b) Im Suchfeld nach *gradle* suchen
   (c) `Gradle Task` auswählen, mit `Open` bestätigen
   (d) In der neuen View sind alle Gradle Projekte aufgelistet
   (e) `gvs-ui > build > build` doppelklicken
   (f) Der Build Task compiliert die Source-Files, lässt Tests laufen und erstellt ein ausführbares JAR im Projekt-Ordner `path/to/project/build/libs`
5. Ausführen von GVS 2.0 UI
   (a) Entweder erstelltes JAR ausführen
   (b) oder in Eclipse: Gradle Task View `application > run` ausführen

Glossar

AWT Abstract Window Toolkit: In die Jahre gekommenes GUI Toolkit zum Erstellen von grafischen Java Applikationen. 12, 43, 113

Dependency Injection Dependency Injection: Abhängigkeiten einer Klasse können zur Laufzeit verändert werden 13, 21, 57, 113

DRY Don't Repeat Yourself: Software Prinzip zur Minimierung von Redundanzen im Code 17, 49, 113

FontAwesome Eine vor allem im Web weit verbreitete Schriftart. Sie basiert auf CSS und LESS und bietet 675 Icons. 39, 41, 88, 113

Force Directed Drawing Algorithm Algorithmus zur dynamischen Positionierung von Nodes mittels anziehenden und abstossenden Kräften. 10, 113

FXML JavaFX XML: XML basierte UI Markup Language für JavaFX Programme. 15, 57, 113

GoF Gang of Four: Sammlung an wiederverwendbaren Software Pattern von Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides 27, 113

GVS Graphs-Visualization-Service: Name der Software die in der vorliegenden Studienarbeit entwickelt wird. I, II, 1, 2, 38, 44, 45, 113

GVS Lib Graphs-Visualization-Service Lib: Komponente, die beim Client in das Software Projekt eingebunden werden muss. Enthält den Socket Client. 4–6, 35, 44, 49, 63, 113

GVS UI Graphs-Visualization-Service UI: Komponente, die für die Visualisierung der Datenstrukturen verantwortlich ist. Enthält den Socket Server. 3, 5, 6, 8, 35, 36, 41, 44, 49, 113

HSR Hochschule für Technik Rapperswil: Fachhochschule in Rapperswil an welcher die Studienarbeit durchgeführt wird. I, II, 1, 113

IDE Integrated Development Environment: Tools zur effizienten Entwicklung von Software Produkten 56, 59, 113

isomorph von gleicher Gestalt oder Struktur 31, 113

JAR Java Archive: Container für ausführbare Java Files 58, 113

JavaFX Framework zur Erstellung von plattformübergreifenden grafischen Anwendungen in Java. Es ist eine komplette Neuentwicklung und seit 2014 der offizielle Nachfolger vom AWT und Swing I, II, 3, 15–17, 26, 27, 57, 61, 113

JMX Java Management Extensions: Werkzeug für die Verwaltung und das Monitoring von Java Applikationen. Erlaubt das dynamische Anpassen von MBeans (Management Beans) 41, 113

MVC Model View Controller: Verbreitetes Softwaremuster zur Trennung der Aufgabenbereiche von Software Komponenten die das UI steuern. 15, 113

MVVM Model View ViewModel: Entwurfsmuster für die Darstellung von modernen UI-Plattformen mittels Databinding. MVVM ist eine Variante von MVC. 14, 15, 18, 113

n-ary Tree beschreibt in der Graph-Theory Bäume in denen jeder Knoten höchstens n Kindknoten besitzt. Binary Trees sind spezialfälle von n-ary Trees für welche n=2 gelten. III, 30, 41, 113

Observer Pattern Observer-Pattern: Das Pattern beschreibt eine Produzenten/Konsumenten Beziehung zwischen zwei Objekten. Der Konsument registriert sich beim Produzenten und wird automatisch notifiziert, wenn beim Produzenten neue Daten vorliegen. 8, 10, 15, 19, 113

POJO Plain Old Java Object: Triviales Java Objekt mit wenigen oder keinen externen Abhängikeiten 15, 23, 25, 113

RUP Rational Unified Process: Vorgehensmodell in der Software Entwicklung, dass ursprünglich von IBM entwickelt wurde. 55, 113

SCRUM Verbreitetes Vorgehensmodell zur agilen Softwareentwicklung. Beinhaltet itertives Vorgehen mit stetigem Kundenfeedback. 55, 113

Single Responsibility Principle Besagt, dass jede Klasse nur eine Aufgabe hat und sämtliche Funktionen einer Klasse sollen zur Erfüllung dieser Aufgabe beitragen. III, 17, 19, 113

Swing GUI Toolkit zur Programmierung von grafischen Benutzeroberflächen. Swing baut auf dem älteren AWT auf. II, 12, 17, 43, 113

Tangle Dependencies von Packages aus tiefen Schichten auf Packages in höher gelegenen Schichten. So zeigt z.B. das Framework SonarQube einen ”Tangle Index” an. 12, 17, 19, 44, 113

Trie auch Präfix-Baum. Eine Datenstruktur zum Suchen von Zeichenketten, welche Datenkompression benutzt, damit mehrmals vorkommende Zeichen nur einmal dargestellt werden. 41, 113

Type Object Teil des Type Object Patterns. Anstelle von Subklassen werden logisch unterschiedliche ”Arten” einer Klasse mit einem Type Object modelliert. e.g. Klasse ”Auto” mit Type Objects "Audi", "VW", ... 19, 113

WPF Windows Presentation Foundation: Klassenbibliothek zur Gestaltung von grafischen Benutzeroberflächen in .NET 15, 113

XML XML: Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten im Format einer Textdatei 5, 6, 8, 113

Literaturverzeichnis

[1] *Atlassian Marketplace: Tempo Timesheets for Jira*. URL: https://marketplace.atlassian.com/plugins/is.origo.jira.tempo-plugin/cloud/overview.

[2] Leipert Buchheim Jünger. "Drawing rooted trees in linear time". In: *Software: Practice and Experiance* (2016). URL: http://onlinelibrary.wiley.com/doi/10.1002/spe.713/pdf.

[3] *Checkstyle*. URL: http://checkstyle.sourceforge.net/.

[4] *Checkstyle: Sun Java Styleguide*. URL: https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/sun_checks.xml.

[5] *Churn Metric*. URL: https://docs.codeclimate.com/v1.0/docs/churn.

[6] *Cognitive Complexity*. URL: https://docs.codeclimate.com/v1.0/docs/cognitive-complexity.

[7] *ControlsFX*. URL: http://fxexperience.com/controlsfx/.

[8] *dom4j*. URL: https://dom4j.github.io/.

[9] *Dropbox*. URL: https://www.dropbox.com.

[10] *EclEmma*. URL: http://www.eclemma.org/.

[11] *Eclipse Buildship*. URL: https://projects.eclipse.org/projects/tools.buildship.

[12] *EGit*. URL: https://eclipse.org/egit.

[13] Anton Epple. *JavaFX 8: Grundlagen und fortgeschrittene Techniken*.

[14] *FindBugs*. URL: http://findbugs.sourceforge.net/.

[15] *FontAwesome*. URL: http://fontawesome.io/.

[16] *Gitflow*. URL: http://nvie.com/posts/a-successful-git-branching-model/.

[17] *Github Repositories*. URL: https://github.com/Graphs-Visualization-Service.

[18] *Gluon Ignite with Google Guice*. URL: http://gluonhq.com/labs/ignite/.

[19] Goldwasser Goodrich Tamassia. *Data Structures & Algorithms in Java*. 2015.

[20] *Google Guice*. URL: https://github.com/google/guice.

[21] *Gradle*. URL: https://gradle.org/.

[22] *GVS UI Code Climate Instanz*. URL: https://codeclimate.com/github/Graphs-Visualization-Service/gvs-ui.

[23] *Jira*. URL: https://project.redbackup.org/projects/GVS/.

[24] *JLS: Java Language Specification, Chapter 6: Names*. URL: https://docs.oracle.com/javase/specs/jls/se9/html/jls-6.html.

[25] *Kraken*. URL: https://de.wikipedia.org/wiki/Kraken.

[26] *Logback*. URL: https://logback.qos.ch/.

[27] *Maintainability Metric*. URL: https://docs.codeclimate.com/v1.0/docs/maintainability.

[28] Thomas J. McCabe. "A complexity measure". In: *IEEE Transactions on Software Engineering* (1976). URL: http://www.literateprogramming.com/mccabe.pdf.

[29] Andreas Egli und Michael Koller. "Graphs- Visualization-Service. GVS Dokumentation". Diplomarbeit. HSR: Hochschule für Technik Rapperswil.

[30] Tilford Reingold. "Tidier Drawings of Trees". In: *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING* (1981). URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1702828.

[31] *Risikomanagement*. URL: https://risikomanager.org/methodenassistent/risikodiagramm-risikograph-risikolandschaft-risikoportfolio-risikomatrix/.

[32] *SLF4J*. URL: https://www.slf4j.org/manual.html.

[33] *Stan4J*. URL: http://stan4j.com/.

[34] *Story Points verständlich erklärt*. URL: http://www.ksimons.de/2011/06/story-points-verstandlich-erklart/.

[35] *Tester Repository*. URL: https://github.com/Graphs-Visualization-Service/gvs-tester.

[36] *The secrets behind story points and agile estimation*. URL: https://www.atlassian.com/agile/estimation.

[37] *Travis-CI*. URL: https://travis-ci.org/.

Abbildungsverzeichnis

Tabellenverzeichnis