|  |
| --- |
|  |
| Graph Visualizer  Eine Applikation zur Visualisierung von Graphen und Algorithmen  **Projekt-Dokumentation** |
| Studiengang: Informatik, Modul BTI7301 (Projekt 1), HS 2013/14  Autor: Patrick Kofmel ([kofmp1@bfh.ch](mailto:kofmp1@bfh.ch))  Betreuer: Jürgen Eckerle ([juergen.eckerle@bfh.ch](mailto:juergen.eckerle@bfh.ch))  Datum: 13. Juni 2014 |

Inhaltsverzeichnis

[1 Einleitung 3](#_Toc390716645)

[2 Requirements und Use Cases 3](#_Toc390716646)

[2.1 User Requirements 3](#_Toc390716647)

[2.2 System Requirements 3](#_Toc390716648)

[2.2.1 Funktionale Requirements 3](#_Toc390716649)

[2.2.2 Nicht-funktionale Requirements 6](#_Toc390716650)

[2.3 Use Cases 7](#_Toc390716651)

[3 Systemarchitektur und Design 7](#_Toc390716652)

[3.1 Schichtenarchitektur 8](#_Toc390716653)

[3.2 Model-View-Controller (MVC) Pattern 8](#_Toc390716654)

[3.3 Package-Struktur 9](#_Toc390716655)

[3.4 Knoten und Kanten 9](#_Toc390716656)

[3.5 Graph-Klassen 9](#_Toc390716657)

[3.6 Algorithmen-Klassen 9](#_Toc390716658)

[3.7 Step-Klassen 10](#_Toc390716659)

[4 Implementation und Tests 10](#_Toc390716660)

[4.1 Coding Conventions 10](#_Toc390716661)

[4.2 Algorithmen-Implementation 10](#_Toc390716662)

[4.2.1 Tiefensuche (DFS) 10](#_Toc390716663)

[4.2.2 Breitensuche (BFS) 10](#_Toc390716664)

[4.2.3 Dijkstra 10](#_Toc390716665)

[4.2.4 Kruskal 10](#_Toc390716666)

[4.3 Tests 10](#_Toc390716667)

[5 User-Dokumentation 10](#_Toc390716668)

[5.1 Programm 10](#_Toc390716669)

[5.2 Graphml-Format 10](#_Toc390716670)

[6 Abbildungsverzeichnis 11](#_Toc390716671)

[7 Tabellenverzeichnis 11](#_Toc390716672)

[8 Glossar 11](#_Toc390716673)

[9 Literaturverzeichnis 11](#_Toc390716674)

[10 Anhang 12](#_Toc390716675)

# Einleitung

Dieses Dokument enthält die Projekt-Dokumentation zur Applikation *Graph Visualizer*. Zu diesem Projekt existiert ein *GitHub-Repository* (Link vgl. Anhang). Dort sind neben dieser Dokumentation auch alle anderen für das Projekt relevanten Artefakte abgelegt (Quellcode-Dateien, API-Dokumentation, UML-Diagramme).

Bei vielen Fachbegriffen wird in diesem Dokument jeweils die englische Form verwendet (z. B. *Requirement* anstelle von *Anforderung*). Eine Liste mit einigen wichtigen Begriffen befindet sich im Glossar.

|  |  |
| --- | --- |
| **Verwendete** Technologien und Tools: | |
| Programmiersprache: | Java (Java SE 7) | |
| Frameworks: | Java Universal Network/Graph Framework (JUNG) | |
| Entwicklungsumgebung: | Eclipse IDE (Kepler Service Release 2) | |
| Build-Management-Tool: | Apache Maven (Maven-Tools Eclipse) | |
| Version-Control-System: | GIT (GIT-Client Eclipse) | |

Tabelle 1: Technologien und Tools

# Requirements und Use Cases

In diesem Kapitel werden zuerst in kurzer Form die *User Requirements* formuliert (basierend auf der Projektbeschreibung des Dozenten). Dann folgt eine Erläuterung der *System Requirements*. Diese beschreiben in detaillierter Form die Systemfunktionen und Systemeigenschaften. Basierend auf den Requirements werden dann die wichtigsten *Use Cases* in einem Diagramm dargestellt.

## User Requirements

Es soll eine Software erstellt werden, welche gerichtete und ungerichtete Graphen erstellen und darstellen kann. Die Software soll es ermöglichen, Graphen aus einer Datei zu laden, zu bearbeiten und in einer Datei zu speichern.

Gleichzeitig soll die Software zur Visualisierung der Traversierung von Graphen dienen. Verschiedene Algorithmen wie z. B. Tiefensuche, Breitensuche, Dijkstra oder Kruskal sollen mit diesem Werkzeug auf einfache Weise visualisierbar werden. Die Visualisierung der Algorithmen soll dabei entweder in einer Animation oder Schritt für Schritt möglich sein. Die Applikation soll sich als didaktisches Hilfsmittel für beliebige Graphen-Algorithmen eignen.

## System Requirements

Die System Requirements bestehen zum grössten Teil aus *funktionalen Requirements*. Diese beschreiben die konkrete Funktionalität der Applikation. Es gibt aber auch einige *nicht-funktionale Requirements*, welche allgemeine Systemeigenschaften beschreiben.

### Funktionale Requirements

|  |
| --- |
| Neuen Graphen erstellen: |
| Gerichtet oder ungerichtet |
| Gewichtet oder ungewichtet (wenn ungewichtet, dann ist das Gewicht bei allen Kanten = 1) |
| Schlingen sind bei gerichteten und ungerichteten Graphen zulässig |
| Mehrfachkanten und Hyperkanten sind *nicht* zulässig |

Tabelle 2: Requirements - Neuen Graphen erstellen

|  |  |
| --- | --- |
| Graph-Elemente (Knoten und Kanten) bearbeiten: | |
| Eindeutigen Name (ID) festlegen und anzeigen | | |
| Knoten: | Neuen Knoten einfügen | |
| Knoten löschen (inzidente Kanten werden automatisch gelöscht) | |
| Höhe und Breite ändern:  Darstellung als Kreis oder Ellipse | |
| Maximal einen Startknoten festlegen (optional):  Darstellung als gestrichelter Kreis oder Ellipse | |
| Maximal einen Endknoten festlegen (optional): Darstellung als gepunkteter Kreis oder Ellipse | |
| Position ändern durch *Drag and Drop* | |
| Kanten: | Neue Kante zwischen zwei Knoten einfügen | |
| Kante löschen | |
| Darstellung als Pfeil (gerichtet) oder Linie (ungerichtet) | |
| Gewicht festlegen und anzeigen | |

Tabelle 3: Requirements - Graph-Elemente bearbeiten

|  |
| --- |
| Graphen bearbeiten: |
| Name und Beschreibung ändern und anzeigen |
| Verschiebung, Drehung, Scherung |

Tabelle 4: Requirements - Graphen bearbeiten

|  |  |
| --- | --- |
| Algorithmus auswählen (Vorberechnung der Animations-Schritte): | |
| Allgemein: | Name und Beschreibung des gewählten Algorithmus anzeigen. | |
| Falls kein Startknoten festgelegt wurde, wähle zufällig einen Startknoten. | |
| Rekursive Tiefensuche (DFS): | Falls ein Endknoten festgelegt wurde und dieser vom Startknoten aus erreichbar ist:  Besuche durch *rekursive Tiefensuche* alle Knoten bis der Endknoten erreicht wurde und speichere die einzelnen Schritte in einer Datenstruktur. | |
| Falls *kein* Endknoten festgelegt wurde oder der Endknoten vom Startknoten aus *nicht* erreichbar ist:  Besuche durch *rekursive Tiefensuche* alle vom Startknoten aus erreichbaren Knoten und speichere die einzelnen Schritte in einer Datenstruktur. | |
| auf gerichtete und ungerichtete Graphen anwendbar | |
| Breitensuche (BFS): | Falls ein Endknoten festgelegt wurde und dieser vom Startknoten aus erreichbar ist:  Besuche durch *Breitensuche* alle Knoten bis der Endknoten erreicht wurde und speichere die einzelnen Schritte in einer Datenstruktur. | |
| Falls *kein* Endknoten festgelegt wurde oder der Endknoten vom Startknoten aus *nicht* erreichbar ist:  Besuche durch *Breitensuche* alle vom Startknoten aus erreichbaren Knoten und speichere die einzelnen Schritte in einer Datenstruktur. | |
| auf gerichtete und ungerichtete Graphen anwendbar | |
| Dijkstra: | Falls ein Endknoten festgelegt wurde und dieser vom Startknoten aus erreichbar ist:  Suche mit dem *Dijkstra-Algorithmus* den kürzesten Weg vom Start- zum Endknoten und speichere die einzelnen Schritte in einer Datenstruktur. | |
| Falls *kein* Endknoten festgelegt wurde oder der Endknoten vom Startknoten aus *nicht* erreichbar ist:  Suche mit dem *Dijkstra-Algorithmus* den kürzesten Weg vom Startknoten zu allen anderen Knoten und speichere die einzelnen Schritte in einer Datenstruktur. | |
| auf gerichtete und ungerichtete Graphen anwendbar | |
| Kruskal: | Berechne mit dem Kruskal-Algorithmus einen minimal aufspannenden Wald und speichere die einzelnen Schritte in einer Datenstruktur. | |
| nur auf ungerichtete Graphen anwendbar | |

Tabelle 5: Requirements - Algorithmus auswählen

|  |  |
| --- | --- |
| Schrittweise Traversierung (mit Hilfe der vorberechneten Datenstruktur): | |
| Visualisierung des Graphen bei jedem Schritt anpassen durch Farbänderung | | |
| Statusmeldung und Zwischenresultate für jeden Schritt anzeigen | | |
| Fortschritt der Traversierung grafisch darstellen (Progressbar) | | |
| Navigationsmöglichkeiten: | Zum Anfang | |
| Zum Ende | |
| Einen Schritt vorwärts | |
| Einen Schritt zurück | |
| Nach dem letzten Schritt: | Falls ein Endknoten festgelegt wurde, zeige den Weg vom Start- zum Endknoten (DFS, BFS, Dijkstra) | |
| Zeige den minimal aufspannenden Wald bei Kruskal | |

Tabelle 6: Requirements - Schrittweise Traversierung

|  |  |
| --- | --- |
| Animation anzeigen: | |
| Abspielgeschwindigkeit: | Zeitintervall festlegen (in Sekunden) | |
| Abspielen: | Animation schrittweise abspielen und GUI sperren | |
| Anhalten: | Animation an aktueller Position anhalten | |
| Abspielung fortsetzen: | Fortsetzung der Animation ab aktueller Position | |
| Abbrechen: | Zum Anfang der Traversierung zurückkehren und GUI aktivieren | |
| Nach dem letzten Schritt: | An aktueller Position anhalten und GUI aktivieren | |

Tabelle 7: Requirements - Animation anzeigen

|  |
| --- |
| IO-Operationen: |
| Graphen aus Datei laden |
| Graphen in Datei speichern |
| Spezielles XML-Format (graphml) wird zur Speicherung verwendet |

Tabelle 8: Requirements - IO-Operationen

### Nicht-funktionale Requirements

|  |  |
| --- | --- |
| Einige nicht-funktionale Requirements: | |
| Erweiterbarkeit: | zusätzliche Algorithmen hinzufügen mit minimalen Änderungen am bestehenden Code | |  |
| Usability: | Shortcuts für alle wichtigen Funktionalitäten | |  |
| Tooltips mit erklärendem Text zu allen wichtigen GUI-Elementen | |  |
| Input-Validierung: | Validierung der XML-Datei beim Laden eines Graphen | |  |
| Validierung von Benutzereingaben | |  |
| Warnhinweise: | Beim Überschreiben einer existierenden Datei | |  |
| Beim Verwerfen nicht gespeicherter Änderungen | |  |

Tabelle 9: Nicht-funktionale Requirements

## Use Cases

Im folgenden Diagramm werden die wichtigsten *Use Cases* dargestellt. Sie zeigen die Systeminteraktionen mit dem User und anderen beteiligten Akteuren.



Abbildung 1: Use Case Diagramm

# Systemarchitektur und Design

In diesem Kapitel folgen einige Erläuterungen zur Systemarchitektur und den verwendeten *Design Patterns*. Für die Systemarchitektur wurden die folgenden *Architectural Patterns* verwendet:

* Schichtenarchitektur
* Model-View-Controller (MVC) Pattern

Es folgt eine Liste mit einigen verwendeten *Design Patterns*:

* (Static) Factory Method
* Composite
* Decorator
* Façade
* Observer
* Iterator
* Strategy
* Command

## Schichtenarchitektur

Die Applikation weisst eine Schichtenarchitektur auf. Jede Schicht bietet Dienste an, die von den darüber liegenden Schichten verwendet werden können. Der Aufbau ist in der folgenden Abbildung dargestellt:



Abbildung 2: Schichtenarchitektur

Das JUNG-Framework stellt viele nützliche Interfaces und Klassen zur Manipulation und Visualisierung von Graphen zur Verfügung. Die Core- und die GUI-Schicht nutzen diese Klassen und Interfaces an sehr vielen Stellen.

Die Core-Schicht bietet der GUI-Schicht grundlegende Dienste an:

* Erzeugung von Instanzen der Datenstruktur *Graph*
* Laden eines Graphs aus einer Datei
* Speichern eines Graphs in einer Datei
* Ausführen eines Algorithmus auf einem Graphen und Rückgabe eines Iterators über alle Animations-Schritte
* Abrufen von Informationen zu bestimmten Algorithmen (Name und Beschreibung)

Die GUI-Schicht ist für die Darstellung des User Interfaces und der Graphen zuständig. Für die Darstellung des Graphen nutzt die GUI-Schicht zum Teil direkt Klassen aus dem JUNG-Framework. Auch alle Event-Handler, die auf User-Interaktionen reagieren, befinden sich in der GUI-Schicht.

## Model-View-Controller (MVC) Pattern

Für die Interaktion des User Interfaces mit den anderen Systemkomponenten wurde das klassische Model-View-Controller (MVC) Pattern verwendet. Die relevanten Klassen für das MVC befinden sich in der GUI-Schicht. Die folgende Abbildung zeigt wichtige Schnittstellen und Klassen von den einzelnen Komponenten des MVC. Es werden dabei nur Assoziations-, Realisierungs- und Vererbungsbeziehungen gezeigt. Die View-Klassen befinden sich auf der rechten Seite, die Controller-Klassen und Interfaces ganz oben und die Model-Klassen und Interfaces unten links.

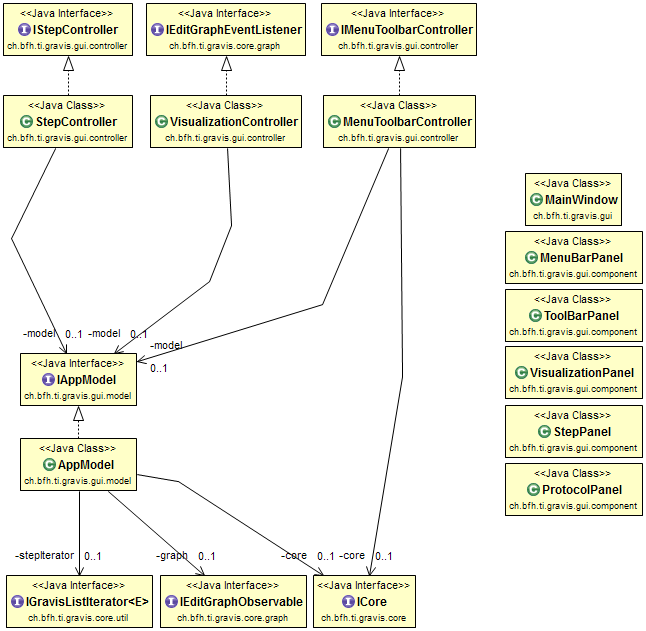


Abbildung 3: MVC-Pattern

Der *View* (Klassen ganz links) hält keine Referenzen zu den Controller- und Modelklassen.

Event-Listener

Observer-Pattern

## Package-Struktur

## Das Core-Interface

Core Klasse zeigen

## Knoten und Kanten

## Graph-Klassen

## Algorithmen-Klassen

## Step-Klassen

# Implementation und Tests

Der Grossteil des Quellcodes wurde selbst implementiert und ist mit Java-Annotationen entsprechend gekennzeichnet. Alle von Drittpersonen übernommenen Klassen sind mit dem entsprechenden Entwickler der Klasse annotiert. Bei einigen aus dem JUNG-Framework abgeleiteten Klassen wurde teilweise bestehender Code aus der Basisklasse übernommen und angepasst. Solche Klassen sind mit dem eigenen Namen sowie mit dem Namen des Basisklassen-Entwicklers annotiert.

## Coding Conventions

## Algorithmen-Implementation

### Tiefensuche (DFS)

### Breitensuche (BFS)

### Dijkstra

### Kruskal

## Tests

# User-Dokumentation

API-Doc Hinweis

## Programm

## Graphml-Format

# Abbildungsverzeichnis

[Abbildung 1: Use Case Diagramm 7](#_Toc390716735)

[Abbildung 2: Schichtenarchitektur 8](#_Toc390716736)

[Abbildung 3: MVC-Pattern 9](#_Toc390716737)

# Tabellenverzeichnis

[Tabelle 1: Technologien und Tools 3](#_Toc390716743)

[Tabelle 2: Requirements - Neuen Graphen erstellen 4](#_Toc390716744)

[Tabelle 3: Requirements - Graph-Elemente bearbeiten 4](#_Toc390716745)

[Tabelle 4: Requirements - Graphen bearbeiten 4](#_Toc390716746)

[Tabelle 5: Requirements - Algorithmus auswählen 5](#_Toc390716747)

[Tabelle 6: Requirements - Schrittweise Traversierung 5](#_Toc390716748)

[Tabelle 7: Requirements - Animation anzeigen 5](#_Toc390716749)

[Tabelle 8: Requirements - IO-Operationen 5](#_Toc390716750)

[Tabelle 9: Nicht-funktionale Requirements 6](#_Toc390716751)

# Glossar

**Auinweon**

Et ut aut isti repuditis qui ium 7

**Batnwpe**

Et ut aut isti repuditis qui ium 9

**Cowoll**

Et ut aut isti repuditis qui ium 11

Mehrfachkante

Hyperkante

Schlinge

User Requirements

System Requirements

Funktionale System Requirements

Nicht-Funktionale System Requirements

Architectural Pattern

Design Pattern

# Literaturverzeichnis

**Literatureintrag**

*Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr* 7

**Literatureintrag**

*Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr* 9

**Literatureintrag**

*Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr* 11

# Anhang

Link zum GitHub-Repository : <https://github.com/kofmp1/GraphVisualizer>

Link zur Projekt-Dokumentation (PDF) im GitHub-Repository: <https://github.com/kofmp1/GraphVisualizer/tree/master/ProjectGraphVisualizer/doc>

Link zur API-Dokumentation: <http://htmlpreview.github.io/?https://github.com/kofmp1/GraphVisualizer/blob/master/ProjectGraphVisualizer/doc/apidocs/index.html>

Link JUNG-Framework

# 