# Designentscheidungen

## Parser/Reader

Die Struktur des Parsers/Readers wurde einem anderen Projekt entnommen in dem ebenfalls das parsen einer Datei nötig war.

Dies ermöglichte es uns einen großen Teil des Aufwands an dieser Stelle zu sparen.

Die einzigen komplett neuen Bestandteile hier waren die verschiedenen Regulären Ausdrücke, welche natürlich an das genutzte Dateiformat angepasst werden mussten.

Bei der weiteren Implementation kam es zu mehr Schwierigkeiten, da die Aufgabe einen hohen Grad an Flexibilität forderte. Dieser führte dazu das Code-Konstrukte notwendig waren, welche den Code unübersichtlich und schwerer zu warten machten. Aus diesem Grund entschieden wir uns eine eigene Subklasse des Graph-Interface zu implementieren, welche es ermöglicht den Parser bei gleicher Funktionalität einfacher zu gestalten(Dazu unter Eigene Erweiterungen mehr).

## Fremdbibliotheken

Um zu entscheiden welche Fremdbibliotheken wir benutzen haben wir uns erst einmal mit den vorgeschlagenen Bibliotheken(JGraphT, JUNG, GraphViz)vertraut gemacht.

Obwohl die Dokumentation von JUNG einstiegsfreundlicher wirkte entschieden wir uns letztendlich doch für JGraphT, da es etablierter ist und somit vermutlich mehr Schnittstellen(Wie z.B. ein JGraph Adapter) für sie entwickelt wurden.

Dies führte ebenfalls dazu, dass wir JGraph anstelle von GraphViz nutzten, da per Adapter eine einfache Einbindung in eine Swing-Oberfläche möglich war.

Des Weiteren ermöglicht uns JGraphT eine interaktive Bearbeitung des Graphen direkt per Input in der Swing Oberfläche welche im weiteren Verlauf implementiert wird.

So wäre es auch möglich simple Veränderungen am Graphen direkt in der Oberfläche vorzunehmen anstelle von Änderungen in der Definitionsdatei.

Solche oder ähnliche Optionen wären uns bei GraphViz komplett entgangen, da diese Bibliothek es nur ermöglicht die Graphen als Bilddatei zu speichern, so wäre jede Form von Bearbeitung der Graphen außerhalb der Definitionsdatei nicht realisierbar und auch die Ergebnisausgabe von Algorithmen wäre nur getrennt von den zugrunde liegenden Graphen möglich.

## Eigene Erweiterungen

Wie zuvor erwähnt entschieden wir uns die JGraphT Bibliothek noch mit einer eigenen Implementation zu erweitern.

Hierbei handelte es sich um 3 Klassen die ich im Weiteren erläutern werde.

1. FlexibleGraph, hierbei handelt es sich um eine eigene Implementation des von JGraphT zur Verfügung gestellten Graph Interface.

Die größte Motivation für eine eigene Implementation war die Tatsache, dass JGraphT zwar Graphen für jeden speziellen Fall, jedoch nicht für einen generellen Fall in dem eine statische Wahl zwischen den verschiedenen Graphentypen(directed/undirected, weighted/unweighted) nicht möglich ist.

So entstand die FlexibleGraph Klasse welche genau in diesem Fall verwendet werden kann, und alle oben genannten Funktionalitäten abhängig von ihren Flags anbietet.

1. Edge, eine einfache Erweiterung der zu JGraphT gehörenden DefaultWeightedEdge.

Die einzige vorgenommene Veränderung betrifft die Visualisierung, welche mit JGraph erstellt wird.

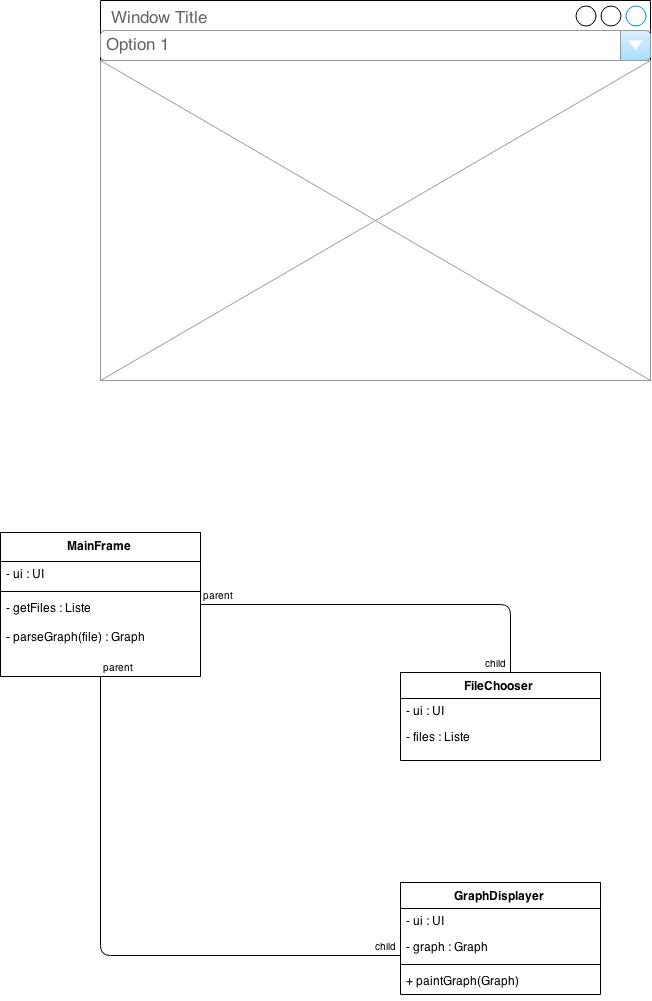
Edge kann auch problemlos in ungewichteten Graphen eingesetzt werden, da die Implementation von JGraphT im ungewichteten Fall jede Kante mit einem Standard Gewicht versieht.

Somit ist es auch möglich Algorithmen welche eine Gewichtung benötigen auf einem ungerichteten Graphen anzuwenden(Natürlich verlieren sie dann an Effizienz).

1. Vertex, die JGraphT Bibliothek wird ohne Standard Knoten ausgeliefert, da im Prinzip jeder Datentyp als Knoten dienen kann.

Abhängig von den Vorgaben entschieden wir uns für eine minimale Implementation welche uns nur genau das ermöglicht was gefragt war, nämlich einen Knoten mit einer ID und einen Attribut zu versehen und diese wenn benötigt weiterzugeben.

## Mock Up



## GUI

Wir haben uns entschieden, die GUI wie im obig dargestellten Mock Up zu erstellen, da dies in Einklang mit Erfahrungen aus PR1 und PR2 steht.

Die Entscheidung Dateien per Dropdownbox und vorgegeben Ordner einzubinden kam aus der Teilaufgabe Dateien zu speichern, da dies später leicht realisiert werden kann.

Eine andere Möglichkeit wäre es gewesen über einen „Laden“ Button ein Systemprompt zur Dateiauswahl zu öffnen, dies ist auch nicht vom Tisch und wird wohl später zusätzlich implementiert.

Das Layout des Graphen ist ein weiterer Punkt der anzusprechen ist.

Wir entschieden uns für ein einfaches „Kreis“-Layout, da dieses es ermöglicht jeden Graphen komfortabel zu lesen.

Einziges Problem bei dieser Darstellung ist das bei größeren Graphen der „Kreis“ aus Knoten sehr groß wird und nicht mehr auf ein Display passt.

Dies wird während der Bearbeitung der nächsten Aufgaben hoffentlich noch verbessert.