Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3314

Visualisierung von Filter/Flow-Graphen auf mobilen Endgeräten

Thomas Ferber

Studiengang:	Softwaretechnik
Prüfer:	Prof. Thomas Ertl
Betreuer:	Dipl. Inf. Florian Haag
begonnen am:	16. April 2012
beendet am:	16. Oktober 2012
CR-Klassifikation:	H.5.2, D.2.2, I.3.6, E.1

Diplomarbeit

Kurzfassung

Filter/Flow-Graphen dienen dem Verständnis und der Vereinfachung von Anfragen an Informationssysteme, indem komplexe Filterausdrücke visualisiert und zugänglich gemacht werden. Hierbei werden einzelne Filterkriterien durch Knoten repräsentiert. Diese Graphen werden bislang nur auf traditionellen Bildschirmen dargestellt. Ziel dieser Diplomarbeit ist es, Filter/Flow-Graphen auf mobilen Endgeräten darzustellen, um Nutzer in die Lage zu versetzen, Abfragen auch unterwegs formulieren zu können oder sich kooperativ in ein vorhandenes Umfeld zur Bearbeitung eines solchen Graphen einklinken zu können. Kleine Bildschirmgrößen von vier Zoll und die ungenaue Touch-Eingabe stellen Herausforderungen bei der Konzeption und Implementierung dar.

Aufbauend auf Darstellungskonzepten für Oberflächen mobiler Endgeräte befasst sich die vorliegende Diplomarbeit damit, ein eigenes Konzept zur Darstellung von Filter/Flow-Graphen auf kleinen Bildschirmen zu entwickeln. Das vorgeschlagene Lösungskonzept bindet gängige Bedienkonzepte mobiler Betriebssysteme mit ein, um eine einfache Bearbeitung der Daten mit maximaler Übersichtlichkeit zu erzielen. Eine Benutzerstudie hat gezeigt, dass das gewählte Overview-Detail-Konzept bei der Orientierung im Graphen und der Bearbeitung von Filterausdrücken überzeugt. Gleichermaßen hat die Studie ergeben, dass die Bediengewohnheiten auf Smartphones großen Einfluss auf das Interaktionserlebnis haben und nicht außer Acht gelassen werden dürfen. Auf Grundlage dieser Ergebnisse kann das Konzept weiter optimiert werden.

Abstract

Filter/flow graphs are used to render queries for information systems simpler and more transparent by visualizing complex filter expressions as well as making these expressions more accessible. In this concept, filter criteria are represented by nodes. Graphs of this type are only displayed on conventional screens as yet. The aim of this diploma thesis is to display filter/flow graphs on mobile devices in order to allow users to formulate requests on the go or to join a cooperative environment for editing a graph of this type. The challenges of drafting and implementing this concept lie in small display sizes of four inches and inaccurate touch input.

Using visualization concepts from user interfaces for mobile devices as a basis, this diploma thesis aims at developing a new concept for visualizing filter/flow graphs on small screens. Established interaction concepts of mobile devices are integrated into the proposed solution in order to facilitate editing data while maintaining clear arrangement. The results of a user survey show that the overview detail concept chosen in this diploma thesis allows easy navigation in graphs as well as filter expression editing. In equal measure, the survey has shown that smartphone operation habits have a great influence on the interaction experience and therefore must be taken into consideration. Based on these results, the concept can be further optimized.

Inhaltsverzeichnis

1 Einleitung

Das vorliegende Kapitel beschreibt die Rahmenbedingungen, unter denen das Dokument entstanden ist, und unter denen die Diplomarbeit durchgeführt wird.

1.1 Motivation

Lineare Datenbankabfragen zur Aufbereitung und Filterung von Daten können heutzutage durch einfache Formulare gut umgesetzt werden. Die Erstellung und Analyse komplexer Filterausdrücke mit alternativen Filterkriterien und Teilausdrücken ist jedoch schwierig. Sie kommen beispielsweise bei Datenbankabfragen an große Informationssysteme vor, sei es in großen Unternehmen, im Internet oder im öffentlichen Verkehr. Sie lassen sich in ihrer textuellen Repräsentation nur schwer in Einzelteile zerlegen, um diese auf Fehler zu prüfen. Zudem birgt die Nutzung textueller Abfragesprachen eine grundsätzliche Fehleranfälligkeit bei der Formulierung komplexer Anfragen.

Auf diesen Erkenntnissen wurden 1991 von Ben Shneiderman so genannte Filter/Flow-Graphen entwickelt, die Filterausdrücke graphisch darstellen [29]. Hierbei wird der Ausdruck anhand einer Datenfluss-Metapher in einen Graphen aus Knoten und Kanten überführt, mit dessen Hilfe sich die Auswertung eines Teil- oder Gesamtausdrucks vereinfachen und aufteilen lässt. Knoten stellen einzelne Filter dar, Kanten den Datenfluss der Ergebnistupel. Der Benutzer kann logische Verknüpfungen direkt anhand der Verknüpfung oder Anordnung der Elemente erkennen, was ihm die Erstellung und Interpretation erleichtert [35]. Aufbauend auf diesem Ansatz wurden neue Konzepte entwickelt, die Filter/Flow-Graphen weiter vereinfachen und deren Darstellungseffizienz durch Ersetzung von Teilgraphen erhöhen [14]. Ein existierendes Konzept, das hierauf aufbaut, wurde für moderne Desktopcomputer mit großen Bildschirmen implementiert.

Des Weiteren ermöglicht die Visualisierung komplexer Datenbankabfragen die Auswertung und Bearbeitung in Zusammenarbeit mit Dritten. Hierbei können Benutzer Teilausdrücke auf einem eigenen Bildschirm oder Bildschirmbereich auswerten. Auf diese Weise kann die Arbeit an komplexen Abfragen auf mehrere Nutzer aufgeteilt werden um so das Verständnis für jeden Teilbereich zu erhöhen. Dies verringert ebenso Engpässe, die durch die Nutzung eines einzelnen Displays in einer Gruppe entstehen würden.

Die Darstellung der Abfragen als Filter/Flow-Graphen eröffnet somit neue Möglichkeiten – bisher werden diese jedoch nur auf traditionellen Bildschirmen genutzt. Mit der Entwicklung kleinerer Displays, wie sie in Smartphones und Tablets eingesetzt werden,

entstand jedoch ein neues Anwendungsfeld für Visualisierungen. Dieses soll genutzt werden, um die Anwendungsmöglichkeiten von Filter/Flow-Graphen zu erweitern und neue Arbeitsmethoden zur Formulierung von Abfragen zu ermöglichen. Die Umsetzung von Filter/Flow-Graphen auf einem Smartphone könnte mobiles Arbeiten an Abfragen ermöglichen. Zudem wäre es möglich, ein Mobilgerät als Client zu verwenden, um mit mehreren Nutzern gemeinsam an einer komplexen Abfrage auf einem großen Display zu arbeiten.

Ziel der Diplomarbeit „Visualisierung von Filter/Flow-Graphen auf mobilen Endgeräten“ ist es, ein Konzept zur Arbeit an solchen Graphen auf mobilen Endgeräten zu entwickeln und als Prototyp umzusetzen. Um an die existierende Filter/Flow-Implementierung des Instituts für Visualisierung und Interaktive Systeme anknüpfen zu können, wird weiterhin die kollaborative Arbeit mit der vorhandenen Desktop-Anwendung berücksichtigt. Da die Umsetzung auf mobilen Endgeräten mit einem kleinen visuellen Arbeitsbereich erfolgt, muss diese speziell hierfür angepasst werden. Um zu zeigen, dass die gewonnenen Erkenntnisse den Erwartungen genügen, umfasst die Entwicklung des Konzepts die Implementierung eines Prototyps, sowie dessen Evaluierung in einer Benutzerstudie. Hierzu wird im Vorfeld eine Recherche zu gängigen Techniken der Graphdarstellung durchgeführt.

Die Diplomarbeit baut bei der Implementierung auf einem Grundgerüst auf, das durch eine Hiwi-Tätigkeit erarbeitet wurde. Dieses stellt das Datenmodell auf dem mobilen Endgerät, sowie die Kommunikationsschnittstelle, bereit. Der angeschlossene Web-Service wird von einer Hilfskraft parallel zur Diplomarbeit gepflegt und weiterentwickelt, so dass der Fokus der Diplomarbeit auf der visuellen Ausarbeitung liegt.

1.2 Kapitelüberblick

Das Dokument wurde gemäß der zeitlichen Umsetzung der Diplomarbeit in die folgenden Kapitel untergliedert:

Kapitel 1, Einleitung Das vorliegende Kapitel. Es beschreibt den Aufbau des Dokumentes und enthält die Motivation hinter der gegebenen Aufgabenstellung.

Kapitel 2, Grundlagen Beschreibt die Grundlagen, die nötig sind, um die Diplomarbeit durchzuführen oder die Arbeitsschritte nachzuvollziehen. Folgende Kapitel bauen auf diesen auf.

Kapitel 3, Zielsetzung Beschreibt die Zielsetzung der Diplomarbeit. Zusätzlich werden die technischen Rahmenbedingungen festgelegt, unter denen diese umzusetzen sind.

Kapitel 4, Themenbezogene Arbeiten/Stand der Forschung Listet themenverwandte Arbeiten auf, die sich mit ähnlichen Problemen oder Teilproblemen wie die vorliegende Diplomarbeit befassen. Die bestehenden Konzepte werden auf Anwendbarkeit untersucht um erste Lösungsansätze festzuhalten und das Lösungskonzept herzuleiten.

Kapitel 5, Lösungskonzept Beschreibt das eigens entwickelte Lösungskonzept und stellt den Kern der Diplomarbeit dar. Alle funktionalen Anforderungen werden anhand von Use-Cases mit Hilfe des entwickelten Konzepts modelliert.

Kapitel 6, Implementierung Beschreibt die technische Umsetzung des Lösungskonzeptes in den Prototyp. Es wird auf die Architektur, sowie Besonderheiten der verwendeten Plattform eingegangen.

Kapitel 7, Evaluierung Enthält Informationen zur Durchführung der Benutzerstudie, eine Beschreibung der Testfälle, sowie die Ergebnisse und deren Auswertung.

Kapitel 8, Zusammenfassung Zusammenfassung, in der ein Fazit gezogen wird. Es wird ein Ausblick auf weiterführende Themen gegeben, die mit der Diplomarbeit zusammenhängen oder sich aus ihr ergeben haben.

1.3 Über das Dokument

Das vorliegende Dokument wurde mit Hilfe des LaTeX-Übersetzers *MikTeX*[1] erstellt. Zur Bearbeitung der Quelldateien wurde vorrangig die plattformunabhängige Anwendung *TexMaker*[2] verwendet. Literaturverweise innerhalb des Dokumentes wurden unter Verwendung von *BibTeX*[3] und dem Referenzenmanager *JabRef*[4] erstellt. Die Kompilierung der `.tex`-Dateien erfolgte unter Zuhilfenahme des Perl-Skripts *Latexmk*[5]. Dieses kompiliert das Skript mit allen notwendigen Zwischenschritten, um alle Verzeichnisse (Inhaltsverzeichnis, Literaturverzeichnis, Glossar) korrekt aufzubauen.

[1] `http://miktex.org/`
[2] `http://www.xm1math.net/texmaker/`
[3] `http://www.bibtex.org/de/`
[4] `http://jabref.sourceforge.net/`
[5] `http://www.phys.psu.edu/~collins/software/latexmk-jcc/`

2 Grundlagen

Dieses Kapitel beschreibt die Grundlagen, die zum Verständnis und der Durchführung der Diplomarbeit nötig sind. Diese sind hierbei in die Themengebiete Graphen, Visualisierung und Mobile Endgeräte untergliedert. In jedem dieser Abschnitte werden Begriffe und Konzepte erklärt, die zum Verständnis der Themenbereiche beitragen und für die Umsetzung der Aufgabenstellung erforderlich sind. Die aufgeführten Grundlagen werden anschließend auf die Aufgabenstellung bezogen.

2.1 Graphen

2.1.1 Graph - Formale Definition und Abgrenzung

Im Folgenden soll ein Graph, wie er im Zusammenhang mit der Arbeit zu verstehen ist, als Grundlage für das verwendete Visualisierungsmodell definiert werden.

Ein Graph $G = (V, E)$ ist ein Zwei-Tupel aus einer Knotenmenge $V = \{v_1...v_n\}$ und einer Kantenmenge $E = \{e_1...e_m\}$, bei dem jede Kante $e \in E$ ein Zwei-Tupel von Knoten darstellt [4]. Jeder Knoten besitzt einen Knotengrad, der über die Anzahl der angeschlossenen Kanten definiert ist [32]. Werden die Kanten in einem Graph als Pfeile statt Linien dargestellt, so spricht man von einem gerichteten Graphen, oder kurz *Digraph*. In gerichteten Graphen ist $e = (u, v) \in V \times V$, in ungerichteten Graphen ist $e = (u, v) \subseteq V$. Im Falle von $u = v$ spricht man von einer Schleife oder einem Zyklus. Falls es zwischen einem Knotenpaar mehrere Kanten geben kann, so nennt man diese Mehrfachkanten. Falls Schleifen und Mehrfachkanten erlaubt sind, so spricht man von einem *Multigraphen*, ansonsten von einem einfachen Graphen. Es wird angenommen, dass es keine Zyklen im darzustellenden Graphen gibt, da diese durch den in Unterabschnitt 2.1.2 definierten Anwendungsfall nicht abgedeckt werden. Man spricht hier auch von einem azyklischen Graphen.

Der Begriff „Graph“ ist weiterhin von der graphischen Darstellung von Werten in Diagrammen, so genannten Funktionsplots, abzugrenzen, die auch unter dem Begriff „Charts“ definiert sind. In dieser Arbeit verstehen wir unter einem Graphen einen einfachen, gerichteten Graphen.

2.1.2 Definition Filter/Flow-Graphen

Filter/Flow-Graphen sind ein Mittel zur visuellen Darstellung von komplexen Datenbankabfragen, so genannten Queries [29]. Der Name leitet sich von einer Metapher zu Wasser ab, das durch eine beliebige Anzahl von Filtern fließt. Die Visualisierung soll das Verständnis von komplexen Abfragen erhöhen, sowie deren Erstellung durch Aufspaltung in kleinere Teilkomponenten erleichtern. Darüber hinaus kann sie die kollaborative Arbeit an Abfragen ermöglichen, beispielsweise zur Informationsgewinnung in größeren Unternehmen. Als Beispiel ist hier die Stuttgarter Straßenbahngesellschaft zu nennen, welche komplexe Abfragen zu Fahrplan und Einsatzdaten leichter bewältigen könnte.

Das Filter/Flow-Konzept geht auf Shneiderman zurück und wurde von ihm erstmals 1991 erwähnt [29]. Es wurde zwei Jahre später (1993) von Degi Young & Ben Shneiderman ausführlich vorgestellt und evaluiert [35]. Die Evaluierung ergab eine deutliche Bevorzugung der graphischen Filter/Flow-Darstellung zur Erstellung von Abfragen gegenüber textbasierten Ansätzen. Es konnte zudem belegt werden, dass die Probanden im Durchschnitt weniger Fehler bei der Erstellung, sowie dem Verständnis von Filter/Flow-Graphen hatten, als bei textuellen Abfragen. Darüber hinaus konnte gezeigt werden, dass die Verwendung von Filter/Flow-Skizzen auch zum Verständnis der textuellen Abfragekonstrukte dienen. Einige Probanden nutzten diese Methode nach dem Erlernen, um ihre textuellen Abfrage innerhalb der vorgestellten Studie zu verifizieren.

2.1.3 Probleme textueller Abfragesprachen

Young und Shneiderman haben in einer Studie herausgefunden, dass Anwender Probleme haben, boolesche Anfragen in einer Abfragesprache zu erstellen [35]. Es wurden viele Versuche unternommen, diesem Problem entgegenzutreten. Unter anderem wurden die Spezifizierung mittels linearem Text in SQL, tabellarische Ansichten und weitere Graphische Implementierungen untersucht, darunter *Venn Diagramms* [19] und *Decision Tables* [13]. Allerdings sind diese Konzepte für unerfahrene Benutzer zu abstrakt und erfordern erheblichen Lernaufwand. Des Weiteren werden sie mit zunehmender Komplexität der Anfragen unhandlicher [29].

Das Problem liegt in der Gewinnung des notwendigen Grundwissens. Der Nutzer muss zum Umsetzen von Anfragen in linearem Text die Bedeutung der booleschen Operatoren kennen, dem logischen „Und“ (der Schnittmenge zweier Bedingungen), des logischen „Oder“ (der Vereinigung zweier Bedingungen) und des logischen „Nicht“ (der Negation einer Bedingung).

Eine weitere Voraussetzung für das Formulieren von SQL-Anfragen ist das Wissen über die Operatorrangfolge (auch *Operatorpräzedenz*). Nutzer haben oft Probleme bei der Klammersetzung, insbesondere verschachtelter Klammern. Tabellarische Ansätze wie QBE (Query By Example) versuchen deshalb, die Verwendung von boolescher Logik zu reduzieren oder mit Wahrheitstabellen ganz zu eliminieren, wie in TEBI (Truth-table Exemplar-Based Interface) [35].

Young & Shneiderman heben in [35] die beobachteten Fehler bei der Verwendung von textuellen Abfragesprachen hervor. In ihrer Studie wurden 20 Probanden mit Aufgaben konfrontiert, die sowohl textuell, als auch mit graphischen Filter/Flow-Elementen zu bearbeiten waren. Folgende Ursachen konnten als Fehlerquellen identifiziert werden, die auch in anderen Studien analysiert wurden:

Vertauschung von „Und“ und „Oder“ Die logischen Operatoren AND und OR wurden häufig falsch interpretiert und bei der Übersetzung von natürlicher Sprache in Textabfragen vertauscht [3], [19]. Dies liegt an der Tatsache, dass Probanden einen Zusammenhang zum alltäglichen, natürlichen Sprachgebrauch herstellten, da hier das „und“ und das „oder“ laut Shneiderman häufig genutzt werden. Dies ist jedoch irreführend.

Natürliche Worte statt Bezeichner Ein weiterer Fehler ist der Einsatz von natürlichen Worten statt der korrekten Bezeichner oder Werte beim Aufbau der Abfrage [26].

Probleme mit Operatorrangfolge Anwender müssen mit der Regel der *Operatorpräzedenz* vertraut sein, um komplexe Queries zu formen. Viele Anwender haben jedoch Probleme bei der Klammersetzung und Verschachtlung von Ausdrücken [3], [19].

Filter/Flow entledigt sich der booleschen Operatoren und Klammern, die laut Studien ein Problem darstellen, indem es sie durch graphische Sequenzen und Abzweigungen darstellt. Die Operatorrangfolge ist implizit durch die Anordnung der Filterobjekte gegeben, siehe Kapitel 4.2.1. In dem verwendeten weiterführenden Konzept werden die Operatoren durch die Kanten selbst definiert, siehe Kapitel 4.2.2.

2.1.4 Erweiterte Filter/Flow-Graphen

Obwohl Filter/Flow-Graphen eine einfache Auswertung der Ergebnismengen durch Verfolgung von Kanten ermöglichen, wirken größere Ausdrücke oft unnötig komplex, da sie redundante Filterausdrücke beinhalten. Durch die folgenden Ersetzungsverfahren können Teilgraphen dieser komplexen Gesamtgraphen jedoch ersetzt und vereinfacht werden, um dem Anwender eine bessere Übersichtlichkeit zu bieten.

Im Originalkonzept von Shneiderman [29] gibt es pro Knoten nur einen eingehenden und einen ausgehenden Flow. Im erweiterten Ansatz kann jeder Knoten eine beliebige Anzahl an Rezeptoren und Emittern besitzen [14]. Jeder dieser Konnektoren kann wiederum an eine einzigartige Menge aus Flows angeschlossen sein. Konnektoren können Filterwerte enthalten, das heißt, jeder Rezeptor und Emitter hat bezüglich der ausgeführten Filteroperation eine andere semantische Bedeutung. Das erweiterte Filter/Flow-Modell wird wie folgt definiert [14]:

Ein erweiterter Filter/Flow-Graph ist definiert durch $\dot{G} = (\dot{V}, \dot{E}, I, O)$. I und O sind die Mengen an Rezeptoren, respektive Emittern. Teilmenge $\mathcal{P}(I) \times \mathcal{P}(O)$ ist die Knotenmenge $\dot{V}$. Jeder Rezeptor und Emitter kann jeweils nur zu einem Knoten gehören. $\dot{E} \subseteq \dot{V} \times \dot{V}$ ist die Kantenmenge. Zwei Knoten v_1 und v_2 sind *direkt verbunden*, wenn

es eine Kante (o, i) gibt, für die gilt, $o \in O_1$, $v_1 = (I_1, O_1)$ und $i \in I_2$, $v_2 = (I_2, O_2)$. Dies kann als $v_1 \rightarrow v_2$ geschrieben werden. Jeder normale Filter/Flow-Graph kann auch als erweiterter Filter/Flow-Graph ausgedrückt werden [14]. Durch diese erweiterte Definition ist es möglich, komplexe Filterausdrücke durch einen einzigen Filterknoten zu repräsentieren, um die Gesamtzahl an Knoten und Kanten zu reduzieren und somit die gesamte Graphstruktur zu komprimieren. Dadurch werden redundante Ausrücke vermieden und die Übersichtlichkeit erhöht. Dies ist vor allem auf dem mobilen Endgerät wünschenswert, da der geringe Bildschirmplatz optimal ausgenutzt werden soll.

Die folgenden Ersetzungsverfahren von Teilgraphen sind möglich [14]:

Bedingte Filter Das Ausführen einer Negation auf einem vorhandenen Filter, um eine Wahr-Falsch-Bedingung zu erzeugen, kann in einem Knoten zusammengefasst werden. Hierbei kann ein Emitter alle Ergebnistupel der Bedingung *Wahr* ausgeben, ein anderer die Tupel für *Falsch*.

Verschachtelte Mengenoperationen Operationen auf Mengen, wie die Eindeutigkeit von Elementen, kann in einem Knoten zusammengefasst werden. Es kann beispielsweise der *distinct*-Operator innerhalb eines Existenzknotens, mittels eines Flags, definiert werden.

Binäre Operatoren Um ein Filterattribut mit mehreren Werten gleichzeitig zu vergleichen, können die Werte, statt in separaten Knoten, in separaten Emittern eines Knotens untergebracht werden.

Between-Filter Filter, bei denen Werte in einem bestimmten Bereich liegen müssen, werden traditionell durch geschachtelte Größenvergleich-Operatoren dargestellt. Diese können nun in einem Knoten mit den entsprechenden Emitterausgängen dargestellt werden.

Abbildung 2.1 zeigt die möglichen Ersetzungsverfahren. Das zu entwickelnde Konzept für mobile Endgeräte verwendet die vorgestellten Ersetzungsverfahren, um die Graphengröße gering zu halten. Im Folgenden bezieht sich der Begriff der „Filter/Flow-Graphen“ deshalb auf den erweiterten Ansatz. Die Darstellung orientiert sich an der Referenzimplementierung für Desktop-Computer, siehe Kapitel 4.2.2.

2.1.5 Filter/Flow-Knotentypen

Ein Filter/Flow-Graph unterstützt die grundlegenden booleschen Operationen *Und*, *Oder* sowie *Nicht*. Diese werden über sequentielle, respektive parallele, Verbindungen zwischen den Filterknoten dargestellt. Das logische *Nicht* wird über Invertierung von ausgewählten Attributen realisiert. Es können jedoch auch weitere Operatoren als Knotentypen modelliert werden. Im Prototyp von Shneiderman 4.2.1 gibt es lediglich Knoten, die einem bestimmten Attribut entsprechen. Im erweiterten Modell, welches für die Implementierung verwendet wird, sind Knoten grundsätzlich als Vergleichsoperatoren definiert, die verschiedene Werttypen annehmen und filtern können. So existieren beispielsweise Vergleichsknoten für Text- oder Zahlenwerte.

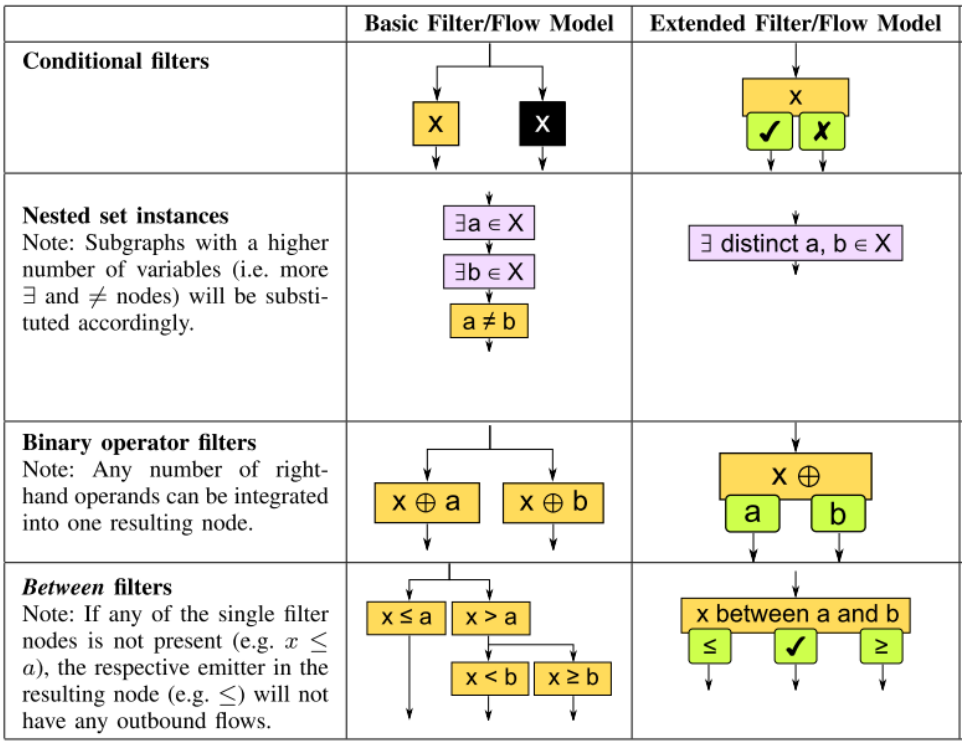


Abbildung 2.1: Mögliche Filter/Flow-Erweiterungen. Links: Original, Rechts: Ersatzknoten

2.2 Visualisierung

Visualisierung, wie sie im technisch-wissenschaftlichen Bereich verwendet wird, ist immer mit einem bestimmten Ziel verbunden [28]. Sie stellt ein Mittel zur Kommunikation zwischen dem Erzeuger der Visualisierung und dem Betrachter dar. Wissenschaftlich-technische Visualisierung hat die Aufgabe, „geeignete visuelle Repräsentationen einer gegebenen Datenmenge zu erzeugen, um damit eine effektive Auswertung zu ermöglichen“ [28]. Sie dient somit der Analyse, dem Verständnis und der Kommunikation von Modellen, Konzepten und Daten.

In dieser Diplomarbeit sollen boolesche Ausdrücke visualisiert werden, um diese benutzerfreundlich darzustellen und ihre Bedeutung möglichst gut vermitteln zu können. Ziel ist es, die Dinge so darzustellen, wie sie tatsächlich vorliegen und den Anwender dabei gleichzeitig in die Lage zu versetzen, Semantiken zu erkennen, verstehen und bewerten zu können [28]. Wissenschaftler sollen dadurch Hilfsmittel erhalten, verborgene Zusammenhänge aufdecken zu können. Visualisierung dient weiterhin zum Austausch von Daten und Arbeitsergebnissen. Visualisierung kann das kollaborative Arbeiten ermöglichen, da visualisierte Ergebnisse für die Zielperson schneller erfassbar sind, als unstrukturierte Daten.

Vorteile graphischer Benutzeroberflächen sind unter anderem, dass Nutzer gestellte Aufgaben schnell und akkurat erledigen können, weniger Frustration zeigen und nicht so schnell ermüden wie bei der Nutzung textbasierter Oberflächen [35], [29].

2.2.1 Visualisierung von Graphen

Visualisierungen von Graphen begegnen uns alltäglich. Das folgende Beispiel aus [4] soll deren Bedeutung veranschaulichen. Als Reisender in einer fremden Stadt bedient man sich gerne der örtlichen öffentlichen Verkehrsmittel. Um sich zu orientieren verwendet man die örtlichen Netzfahrpläne, wie beispielsweise in Abbildung 2.2, die einem auch ohne Kenntnisse der örtlichen Sprache die Routenfindung ermöglichen. Die Verkehrsnetze sind hierbei jedoch keinesfalls geographisch korrekt auf den bekannten Netzfahrplänen abgebildet, wie in Abbildung 2.3 zu sehen. Wäre dies der Fall, so würde es der Mehrheit der Benutzer sicherlich schwieriger fallen, ihre Route einfach und schnell planen zu können.

In beiden Fällen werden Routen farblich kodiert, gleiche Routen in der gleichen Farbe, sowie die Stationen mit einer speziellen Form. Der Unterschied liegt im verwendeten Layout. In der schematischen Ansicht (Abbildung 2.2), wie sie uns heutzutage bekannt ist, wurden nur horizontale, vertikale und diagonale Linien verwendet, um das Verkehrsnetz abzubilden. Dies dient zur Reduzierung der visuellen Komplexität, wie sie in Abbildung 2.3 zu finden ist. Dennoch repräsentiert die Karte die geographischen Verhältnisse relativ genau. Jedoch sind Distanzen von einer Station zur nächsten nur quantitativ abgebildet. Verkehrsteilnehmer zählen die Distanz von einer Station zur nächsten nicht in Kilometern,

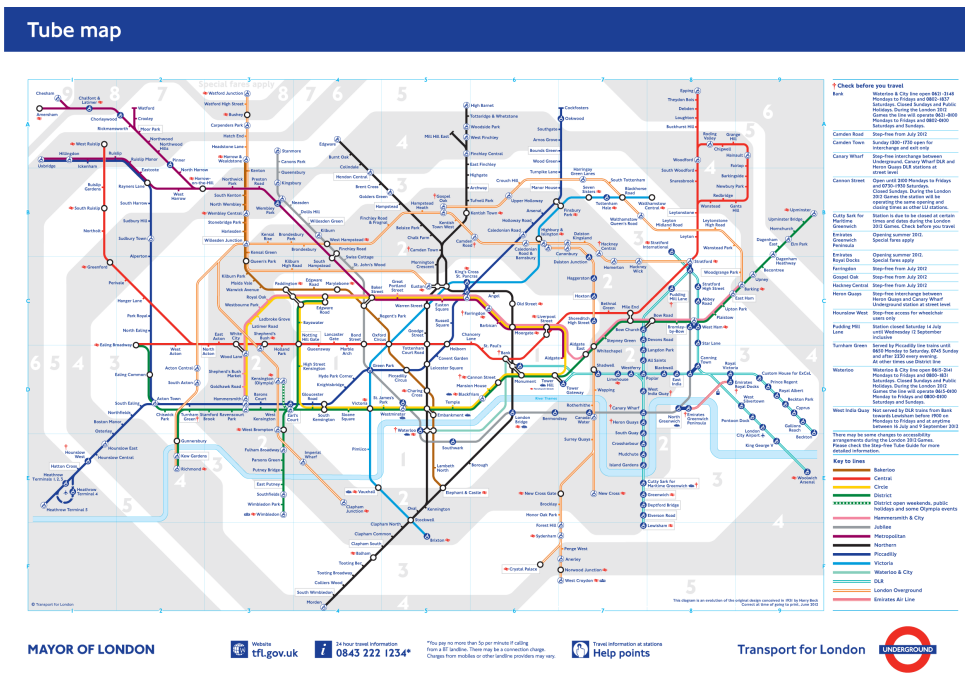


Abbildung 2.2: Karte der Londoner U-Bahn, wie sie seit Jahren im Einsatz ist. [a]

[a] http://www.tfl.gov.uk/assets/downloads/standard-tube-map.gif

sondern in der Anzahl der Zwischenstopps. Somit sind gleichgroße Liniensegmente zwischen Stationen ausreichend, um dem Benutzer die notwendige Information mitzuteilen. Zudem wird das Netzzentrum vergrößert dargestellt, um eine Ansicht in einer separaten Karte zu vermeiden.

Mit diesen einfachen graphischen Mitteln konzentrieren sich Netzkarten auf die Übermittlung notwendiger Informationen an den Nutzer. Ziel von Netzplänen ist es, dem Nutzer die Verbindungen zwischen den einzelnen Stationen über die verfügbaren Linien klar zu vermitteln. Somit sind die verwendeten Karten aus den Abbildungen 2.2 und 2.3 Beispiele von Graphenvisualisierung, deren Nützlichkeit über ihr Layout definiert wird.

Ein weiteres Beispiel findet sich in [1]. Der Artikel beschreibt sowohl die zuvor präsentierte Londoner Subway Map als auch neue Versionen der New Yorker U-Bahn-Netzpläne. Eddie Jabbour, Grafik Designer for Kick Design, fand die offizielle Karte der New Yorker U-Bahn-Linien zu verwirrend und unklar. Er setzte daraufhin eine abstraktere Version mit einem neuen Layout auf, das in Abbildung 2.4 zu sehen ist. Zu den Änderungen zählen insbesondere:

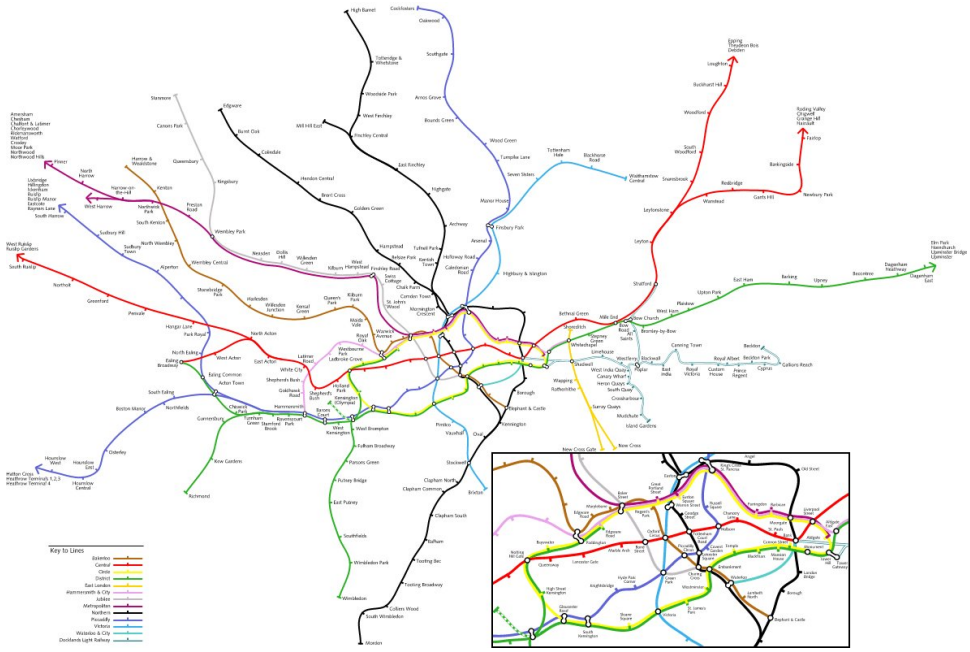


Abbildung 2.3: Geographisch korrekte Karte der Londoner U-Bahn Linien. [a]

[a] http://yellowcardesign.com/blog/wp-content/uploads/2010/01/geographical_map.jpg

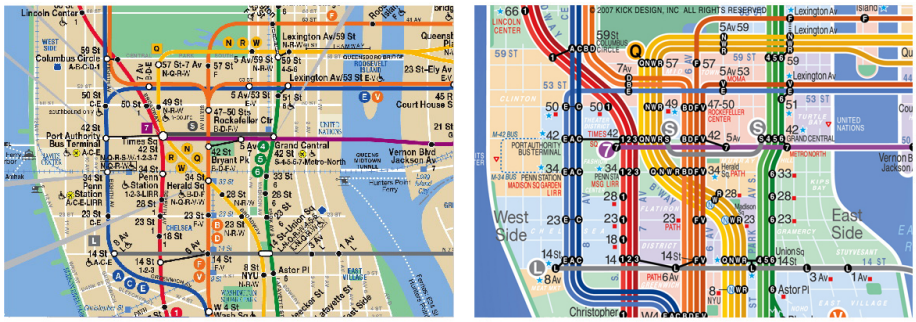


Abbildung 2.4: Vergleich des original U-Bahn Netzplans von Manhatten mit der abstrahierten Version von Eddie Jabbour. [1]

- Eine reine Präsentation der U-Bahn-Linien ohne Buslinien wie es in der offiziellen Version der Fall ist. Das U-Bahn-System sei für sich komplex genug um auf einer Karte allein repräsentiert zu werden.
- Linien verlaufen parallel zueinander, was sie ungenauer aber besser lesbar macht. Ebenso werden Linien nur noch horizontal, vertikal und diagonal dargestellt, wie es in der Londoner Subway Map der Fall ist. Die so erzeugte Gitterstruktur hilft vor allem Fremden bei der Orientierung im Verkehrsnetz.
- Das Clustering von Zügen einer Linie entfällt. Somit sind mehrere parallel verlaufende Linien sichtbar, die jedoch eindeutig gekennzeichnet und verfolgbar sind.
- Wörter benutzen alle dieselbe Schrift und verlaufen horizontal statt diagonal.

Die Beispiele zeigen, dass eine Veränderung des Layouts durch eine Reduzierung von Informationen eine bessere Übersicht schaffen und dem Nutzer der Visualisierung die notwendigen Informationen schneller vermitteln kann. Auf die vorliegende Aufgabenstellung bezogen bedeutet dies, dass ein Layout gefunden werden soll, das eine akkurate, aber vor allem übersichtliche Darstellung des anzuzeigenden Graphen ermöglicht. Dies kann durch Reduktion von Information, Hinzufügen semantischer Informationen oder einer veränderten Darstellung der Grundelemente erreicht werden.

2.2.2 Graph Komprimierungstechniken

Die folgenden Techniken können verwendet werden, um große Graphen visuell zu komprimieren:

Fisheye

Beim Fisheye-Ansatz geht man von einer Übersicht des gesamten Graphen aus. Jedoch wählt der Benutzer einen Fokus-Knoten zum Mittelpunkt der Ansicht, von dem aus die Darstellung des umgebenden Kontext berechnet wird. Nahegelegene Informationen werden hierbei normal dargestellt, entfernte Informationen werden entsprechend einer Funktion auf ihrer Distanz verkleinert. Es entsteht eine Ansicht, die dem bekannten Fisheye-Effekt gleicht. Durch diese Komprimierung kann eine Übersicht großer Graphen, bei gleichzeitiger Betrachtung eines wichtigen Teilbereichs, dargestellt werden.

Das Fisheye-Layout eignet sich nur begrenzt für die Anwendung auf Mobilgeräten, da die benötigte Gesamtübersicht aus den gleichen Gründen wie in Kapitel 4.1.1 beschrieben nicht gegeben ist. Des Weiteren wird die Struktur des Graphen zerstört, die bei Filter/Flow jedoch relevant ist.

Clustering

Beim Clustering wird die Übersicht des Graphen hierarchisch neu berechnet. Zusammengehörende Bereiche werden anhand definierter Funktionen in so genannten Clustern zusammengefasst, welche in die Gesamtübersicht eingebettet sind. Somit lässt sich eine komprimierte Übersicht generieren, die auf kleineren Bildschirmen betrachtet werden kann. Die Inhalte der einzelnen Cluster können auf Wunsch betrachtet und bearbeitet werden.

Der Ansatz des Clusterings eignet sich aufgrund der einstellbaren Clustergröße zur Informationsreduktion gut für die Umsetzung auf einem Mobilgerät. Sie wird bereits im Filter/Flow-Prototyp von Young und Shneiderman eingesetzt (siehe Kapitel 4.2.1).

2.2.3 Probleme bei der Portierung gängiger Graph-Visualisierungskonzepte

Die genannten Standardkonzepte zur Visualisierung von Graphen bergen bei der direkten Portierung auf ein mobiles Endgerät die folgenden Probleme, nach Relevanz sortiert:

Bildschirmknappheit Der Bildschirm eines Mobilgerätes ist mit durchschnittlich 4" zu klein, um den kompletten Graphen, mitsamt seinen Informationen, darzustellen. Detailinformationen wären nicht mehr erkennbar, bei gleichzeitiger Darstellung der kompletten Struktur.

Umständliche Navigation Da ein Mobilgerät über einen berührungsempfindlichen Bildschirm gesteuert wird, wird oft ein Pan- und Zoom-Konzept für die Bedienung mit einem oder zwei Fingern implementiert. Dies ist für große Graphen ungeeignet, da man leicht die Orientierung verlieren kann. Es wird die Annahme gestellt, dass ein Konzept aus Zoomen und Verschieben der Ansicht den Benutzer nur schwer an ein gewähltes Ziel bringt, um dieses letztendlich zu bearbeiten. Die Hypothese wird in der Benutzerstudie durch ein Mockup evaluiert.

Bedienprobleme Durch den berührungsempfindlichen Bildschirm ist es nicht möglich, Bedienelemente beliebig klein zu gestalten. Durch die Bedienung mit den Fingern, meist dem Daumen, ergeben sich natürliche Mindestgrößen von 9,2 bis 9,6 Millimetern bei der Darstellung der Elemente [22]. Es müssen alternative Bedienkonzepte untersucht werden, die an die Bedienung über den Touchscreen angepasst sind.

Ressourcenknappheit Der Anwendungsspeicher mobiler Endgeräte ist stark begrenzt. Um diesem Punkt zu begegnen wird ein ausreichend leistungsfähiges Mobilgerät als Entwicklungsgerät ausgewählt, damit Engpässe bei der Implementierung vermieden werden können.

Den erarbeiteten Problemstellungen soll nun durch Entwicklung eines geeigneten Konzeptes begegnet werden. Dieses soll die Bedienung in den genannten Punkten verbessern.

2.2.4 Visualisierung von Graphen auf kleinen Displays

Die meisten Konzepte zur Visualisierung von großen Graphen bedienen sich einer ganzheitlichen Übersicht [15]. Diese wird durch platzeffiziente Layouts, Clustering-Techniken oder Matrix-Darstellungen erreicht. Aufgrund der erwähnten Einschränkungen, ist dies jedoch in diesem Fall nicht möglich. Es ist somit ein Verfahren notwendig, welches den Fokus auf einen Teilbereich des Graphen legt, um eine bestimmte Aufgabe umzusetzen. Diese komplexitätsreduzierenden Verfahrens werden in die drei folgenden Kategorien unterteilt [15]:

Attributbasierte Abstraktion Diese Verfahren nutzen die Attributwerte der Knoten, um die Komplexität zu reduzieren. Knoten mit gleichen Attributen werden gebündelt, um Dimensionen im Graphen zu reduzieren. Allerdings gehen Informationen einzelner Kanten bestimmter Knoten verloren. Das Modell eignet sich somit nicht für den Einsatzfall, da die Filter/Flow-Metapher durch die Bündelung verloren geht.

Kontextbezogene Ansichten In diesen Systemen wählt der Nutzer einen Knoten als Fokus aus, um dessen Umgebung (Kontext) zu betrachten. Das Konzept eignet sich für Knoten mit mittleren Knotengraden.

Berechnungsgestützte Ansätze Diese Verfahren sind stark rechnergestützt und setzen die Berechnung von Metriken auf den Graphen voraus.

Diese Ansätze verändern jedoch durch Komprimierung die Struktur des Graphen, da sie für sehr große Graphen konzipiert wurden. Es soll ein Verfahren gefunden werden, das die Struktur intakt lässt und eine möglichst effiziente Bedienung liefert.

Für die Darstellung eines Filter/Flow Graphen ist die Übersichtlichkeit, die visuelle Anordnung der Knotenelemente, sowie deren leichtes Verständnis entscheidend, um einen Vorteil für unerfahrene Nutzer zu bieten. Die Anordnung der Knoten an einem Gitter, der prototypischen Implementierung von Degi Young (siehe Kapitel 4.2.1) ermöglicht eine einfache Assoziation mit der Schnittmenge und der Vereinigung zweier Mengen. Im

vorliegenden Fall werden diese Mengenoperationen durch die Verbindung mit Kanten dargestellt, da von der Hostapplikation eine freie Anordnung der Elemente vorgesehen ist. Diese muss bestmöglich auf dem kleinen Bildschirm des Endgerätes abgebildet werden können, ohne die Assoziation mit den Mengenoperationen zu verlieren.

Für die Aufgabenstellung muss die vollständige Funktionalität der Hostapplikation abgebildet werden, wie sie in Kapitel 3 aufgelistet ist. Knoteninhalte müssen vollständig editierbar sein. Hierbei geraten die aufgelisteten Konzepte schnell an ihre Grenzen, da sie Details abstrahieren und die Bedienung somit beeinträchtigen.

2.3 Mobile Endgeräte

In dieser Arbeit sollen die aufgeführten Themenbereiche Graphen (Kapitel 2.1) und Visualisierung (Kapitel 2.2) verknüpft und auf mobilen Endgeräten umgesetzt werden. Hierfür ist ein Verständnis von Mobilen Endgeräten notwendig, wie sie in Zusammenhang mit dieser Diplomarbeit verstanden werden. Als Mobiles Endgerät kann allgemein jedes portable Computing-Gerät bezeichnet werden. Hierunter fallen unter anderem Laptops, Tablets sowie Smartphones. Die vorliegende Arbeit beschäftigt sich jedoch ausschließlich mit mobilen Endgeräten mit kleinen Bildschirmen und kapazitiver Eingabefunktion - den Smartphones. Diese sollen im Folgenden definiert werden.

2.3.1 Definition Smartphone

Als Smartphone bezeichnet man laut [7] moderne Mobiltelefone mit einem Touchscreen zur Anzeige sowie Eingabe und vielfältigen multimedialen Fähigkeiten. Über den Touchscreen können gleichzeitig Daten angezeigt und bearbeitet werden. Heutige Smartphones vereinen die Funktionalitäten mehrerer Geräte in einem, darunter Handy, Organizer, Navigationsgerät, MP3-Player, Spielekonsole sowie Tablet-PC. Tabelle 2.1 vergleicht die Funktionen und Vorteile von Smartphones gegenüber älteren Mobilfunkgeräten.

Smartphones werden mit einem vollwertigen mobilen Betriebssystem betrieben. Diese bieten, analog zu Desktop-Betriebssystemen, neue Stärken und Schwächen gegenüber simpleren Handy-Betriebssystemen. Die folgenden fünf Betriebssysteme sind in diese Kategorie einzuordnen:

- Android von Google
- iOS von Apple
- Windows Phone 7 von Microsoft
- webOS von HP
- Blackberry OS von RIM

Klassisches Handy	Smartphone
Hardwaretasten: Große Tasten, kleines Display	Touchscreen: Mehr Nutzfläche durch Verschmelzung von Display und Eingabe
WAP für langsame Internetverbindungen	Mobiles Internet: Breitbandzugang mit integriertem Web-Browser
Einfacher Kalender	Organizer-Funktionalität
Externe Antenne	Integrierte Antenne
Sperriges Design aufgrund von Hardwarerestriktionen	Flaches, kompaktes Design
Fester Funktionsumfang	Installation von Zusatzfunktionalität durch Apps
Telefonie im Vordergrund	Multimediafunktionalität, audiovisuelle Inhalte

Tabelle 2.1: Klassische Handys und Smartphones im Vergleich

Für die Diplomarbeit wurde die Plattform Android zur Implementierung des Prototyps ausgewählt. Die Auswahl erfolgte, unter anderem, anhand der offenen Entwicklungsumgebung, welche von Google als *Android SDK* bereitgestellt wird.

2.3.2 Interaktion mit Smartphones/Navigation

Bei den verbreiteten Smartphone-Betriebssystemen haben sich die folgenden Konzepte durchgesetzt, um Inhalte auf den kleinen Bildschirmen zu präsentieren und dem Benutzer die Möglichkeit zu geben, zwischen diesen zu navigieren:

Sichtenkonzept Das Sichtenkonzept wird von jedem gängigen Smartphone-System unterstützt. Es ermöglicht die Bildschirmfüllende Anzeige einer Anwendung oder Aktivität.

Stapelnavigation Die Navigation zwischen verschiedenen Sichten erfolgt meist nach dem „First-In-First-Out“-Prinzip eines Stapels. Sichten, die eine Verfeinerung der Darstellungsebene bieten, werden auf den Stapel geschoben. Möchte der Nutzer zurück navigieren, so drückt er eine dafür vorgesehene Navigationstaste und gelangt zur vorigen Ansicht zurück. Dies bewirkt, dass die derzeitige Ansicht wieder vom Stapel genommen wird.

Navigation über Tabs Um dem Nutzer einen schnellen Wechsel zwischen gleichwertigen Ansichten zu gewähren, kann eine Tab-Leiste verwendet werden. Hierbei kann jede Ansicht als ein Tab dargestellt werden. Der Bildschirmbereich verkleinert sich dabei um die Höhe der Tab-Leiste, der Rest der Anzeige kann von der gewählten Ansicht eingenommen werden.

Touch-Bedienung Da nahezu jedes moderne Smartphone über einen berührungsempfindlichen Bildschirm verfügt, werden Aktionen generell über das Berühren des Bildschirms ausgeführt. Elemente müssen hierbei eine Mindestgröße von 9,2 bis 9,6 Millimeter vorweisen, um vom Benutzer entsprechend gut bedienbar zu sein [22].

Orientierungsabhängige Darstellung Aufgrund der Tatsache, dass sich mobile Displays in jede beliebige Richtung drehen und betrachten lassen, ist es gängig, den Inhalt entsprechend der Orientierung des Gerätes anzupassen. Somit können breite Inhalte besser dargestellt werden, wenn der Benutzer das Smartphone in eine Landscape-Orientierung dreht.

2.3.3 Inhaltsbetrachtung auf kleinen Bildschirmen

Die Betrachtung von Inhalten auf kleinen Bildschirmen, wie sie in Smartphones zu finden sind, wird immer wichtiger. Besonders das Betrachten von Webseiten über Smartphones soll zunehmen, da vor allem in Entwicklungsländern das Smartphone für viele Menschen meist die einzige Anbindung an das Internet darstellt [27]. Die Herausforderung besteht darin, einen Kompromiss zwischen einer guten Lesbarkeit des Inhalts und einer ansprechenden Visualisierung der Elemente zu erreichen – jedoch unter Beibehaltung der Originalstruktur, um eine einfache Navigation zu ermöglichen. Viele Ansätze, die eine gute Lesbarkeit des Inhalts erzielen, wie das beschriebene „Narrow Layout“ einer Webseite, zerstören die Struktur des Originals und machen eine räumliche Navigation unmöglich. Somit geht die Bedeutung, die durch die Struktur ausgedrückt wird, verloren.

Dieses Problem ist auf die Darstellung von Filter/Flow-Graphen im vorliegenden Falle übertragbar. Die Struktur des Graphen muss erhalten bleiben, um eine räumliche Navigation und Orientierung zu ermöglichen. Dies rührt vor allem von dem Ansatz zur kooperativen Arbeit her, der verfolgt wird. Gleichzeitig muss jedoch eine gute Lesbarkeit und Bedienbarkeit sichergestellt sein, um die Daten, in diesem Falle Knoten und Kanten, editieren zu können.

Als Ausgangssituation für die Betrachtung von Webseiten existieren zwei naive Layout-Ansätze [27]. Das Originallayout unter Zuhilfenahme einer Zoomfunktion, um eine Übersicht zu erhalten und einen Teilbereich vergrößert zu betrachten, sowie ein kompaktes „Narrow Layout“, das den Inhalt strikt auf die Bildschirmbreite beschränkt und nur vertikale Navigation ermöglicht.

Das Originallayout zeigt den Inhalt, wie er ursprünglich angezeigt wird, hier entsprechend der Hostapplikation. Die Struktur bleibt erhalten und ist dem Anwender somit vertraut, die Navigation fällt leicht. Als Nachteile nennen die Autoren:

1. Text wird nicht umgebrochen und ist schwierig zu erfassen, wenn er über die Bildschirmbreite hinausragt. Ebenso schwierig zu lesen ist Text, der durch eine Zoom-Funktion zu klein dargestellt wird.

2. Der Bildschirmbereich wird nicht bestmöglich ausgenutzt. Ungenutzte Räume zwischen Objekten erschweren die Navigation und geben dem Benutzer das Gefühl, die Orientierung zu verlieren.

Die Probleme sind auf die Darstellung von Filter/Flow-Graphen übertragbar. Detaillierte Knoteninhalte können bei vollständiger Graphenübersicht nicht groß genug angezeigt werden, um sie lesbar zu machen. Dies gilt vor allem bei Graphen ab einer mittleren Größe. Existieren Knoten, die weit auseinanderliegen, so ist die Darstellung der Originalstruktur ohne eine Komprimierung ebenfalls ungeeignet.

Das zweite naive Darstellungskonzept, das „Narrow Layout“ bietet die Vorteile, dass Text immer einfach zu lesen ist und der Inhalt kompakt dargestellt wird, ohne leere Zwischenräume, die die Navigation erschweren. Bei diesem Ansatz, basierend auf Komprimierung, bestehen jedoch die folgenden Probleme [27]:

1. Inhalte, die auf eine bestimmte Breite ausgelegt wurden, werden verzerrt und sind nach der Skalierung nicht mehr lesbar. Bei Tabellen kann es vorkommen, dass durch Verschiebung die Bedeutung ganz verloren geht.
2. Der Benutzer kann nicht anhand der Position eines Inhaltes navigieren, da er nicht weiß, wo der Inhalt im kompakten Layout erscheint. Somit ist es ebenfalls schwer, die Bedeutung des angezeigten Inhaltes zu erfassen, während der Nutzer durch den Inhalt scrollt, da er dessen Position nicht dem Originallayout zuordnen kann.
3. Kompakte Layouts erschweren die Transition des Inhaltes von einem Zustand in den nächsten, da der Benutzer nicht direkt sieht, welcher Teil des Inhalts sich geändert hat. Ohne eine Übersicht, in der dies erkennbar ist, muss er deshalb blind zu der aktualisierten Stelle navigieren. Dabei kann es passieren, dass der Nutzer gar nicht merkt, dass ein Zustandswechsel stattgefunden hat und er eine Aktion versehentlich mehrmals ausführt, wie beispielsweise das Anklicken eines Links [27].

Diese Probleme sind auf die Darstellung eines Filter/Flow-Graphen übertragbar. Besonders der zweite Punkt macht ein Konzept basierend auf Komprimierung unbrauchbar. Der Benutzer muss stets wissen, wo er sich im Graphen befindet, damit die kooperative Arbeit mit der Hostapplikation möglich ist.

2.4 Google Android

Android stellt ein offenes Betriebssystem für Smartphones und Tablets dar. Es befindet sich zum Zeitpunkt der Diplomarbeit in Version 4.1 mit Codenamen „Jelly Bean“. Android wurde in Java programmiert, der Quelltext ist frei verfügbar und kann online [1] eingesehen werden. Android wurde speziell für die Bedienung mittels Touchscreen ausgelegt. Das *Android SDK* bietet zahlreiche APIs zur Programmierung eigener Applikationen, kurz

[1] `http://source.android.com/`

„Apps“, die über den offiziellen *Play Store*[2] vertrieben werden können. Die Programmierung dieser Apps erfolgt grundsätzlich ebenfalls in Java. Alternativ kann das „Mono for Android-Framework“ der Firma Xamarin verwendet werden, siehe Kapitel 6.2.

2.4.1 Benutzerschnittstellen & Bedienkonzept

Das Betriebssystem Android ist für Smartphones und Tablets mit begrenzten Displaygrößen ausgelegt. Es existiert somit kein Fensterkonzept, bei dem mehrere Anwendungen gleichzeitig auf dem Bildschirm sichtbar sind, so wie man es von Desktop-Betriebssystemen kennt. Eine Applikation nimmt bei der Ausführung auf einem Smartphone somit den gesamten sichtbaren Bildschirmplatz ein. Die folgenden Kapitel erklären, wie Android es dennoch ermöglicht, mehrere Applikationen und Programmteile gleichzeitig auszuführen. Des Weiteren wird auf das Sichtenkonzept eingegangen, was zum modularen Aufbau von Benutzeroberflächen dient.

2.4.2 Der Activity Lifecycle

Activities stellen den Kern der Benutzer-System-Interaktion dar. Eine Activity ist laut [9] eine Komponente, die eine Bildschirmansicht zur Verfügung stellt, mit der der Benutzer interagieren kann und es ihm ermöglicht, bestimmte Aufgaben durchzuführen, wie beispielsweise das Wählen einer Telefonnummer, das Senden einer Email oder die Aufnahme eines Fotos. Jede Activity verfügt hierbei über ein Fenster, das grundsätzlich den Bildschirm komplett ausfüllt.

Eine App besteht normalerweise aus mehreren Activities, welche lose gekoppelt sind und sich gegenseitig aufrufen können. Eine Activity wird hierbei als „Main-Activity“ gekennzeichnet und stellt den Einstiegspunkt in das Programm dar. Der Benutzer sieht diese Activity, wenn die App zum ersten mal gestartet wird und kann von ihr aus weiter navigieren. Wenn eine neue Activity aufgerufen wird, wird diese angezeigt und erhält vom System den Fokus. Die zuvor geöffnete Activity wird hierbei angehalten und auf einen Stack geschoben, dem so genannten „Back Stack“. Dieser dient dem Nutzer bei der Rückwärtsnavigation durch seine getätigten Aufrufe. Der Benutzer kann hierbei bis zum Wurzelelement, dem Homescreen, zurück navigieren.

Jede Activity unterliegt hierbei dem Activity Lifecycle, bei dem verschiedene Zustände der Activity durchlaufen werden. Da sich wegen des bildschirmfüllenden Fensters stets nur eine Activity im Vordergrund befinden kann, muss Android in der Lage sein, Activities pausieren und wiederherstellen zu können. Hierfür wird die Activity vom System über die so genannten „Activity Callback Methods“ darüber informiert, in welchen Zustand sie als nächstes wechselt. Dies umfasst beispielsweise das Starten und Stoppen der Activity, sowie die Wiederaufnahme oder das Zerstören dieser. Durch das Überschreiben der

[2] `https://play.google.com/store?hl=de`

Callback-Methoden kann der Programmierer Methoden aufrufen, die für den Übergang in den jeweiligen Zustand angemessen sind.

Im Folgenden findet sich eine Auflistung aller Lifecycle-Callback-Methoden, auf die eine Activity reagieren kann, wie sie in [9] beschrieben ist.

`onCreate()` Wird aufgerufen, wenn die Activity das erste Mal erstellt wird. Hier sollte das Objekt, sowie zugehörige Sichten, initialisiert werden.

`onRestart()` Wird aufgerufen, wenn die Activity gestoppt wurde, kurz bevor sie wieder gestartet wird.

`onStart()` Wird aufgerufen, bevor die Activity dem Benutzer sichtbar gemacht und präsentiert wird.

`onResume()` Wird aufgerufen, kurz bevor der Benutzer mit der Activity interagieren kann. Dies stellt den höchsten Punkt im Activity Stack dar.

`onPause()` Wird aufgerufen, wenn das System dabei ist eine andere Activity wiederaufzunehmen. An dieser Stelle können nicht gesicherte Änderungen dauerhaft gespeichert werden, damit diese nicht verloren gehen. Das System wartet mit der Aufnahme der neuen Activity, bis die Methode behandelt wurde.

`onStop()` Wird aufgerufen, wenn die Activity nicht länger für den Benutzer sichtbar ist. Dies kann passieren, wenn eine andere Activity den Fokus erhält.

`onDestroy()` Wird aufgerufen, bevor die Activity zerstört wird und stellt somit den letzten Callback an diese dar. Dies kann durch einen Aufruf an `finish()` ausgelöst werden, oder durch das System um kurzfristig Speicherplatz freizugeben. Der Programmierer hat die Möglichkeit über `isFinishing()` auf diese beiden Fälle zu reagieren.

Abbildung 2.5 veranschaulicht die Reihenfolge, in der die einzelnen Methoden vom System aufgerufen werden.

2.4.3 Das Sichtenkonzept von Android

In Kapitel 2.4.2 wurden Activities als Kern-Interaktionsklassen dargestellt. Der vorliegende Abschnitt beschreibt, wie die Benutzeroberfläche einer Activity aufgebaut ist und wie Android diese rendert.

Innerhalb des Fensters einer Activity existiert eine Hierarchie aus so genannten Views, den Datenansichten und Bedienelementen des Android Frameworks. Eine Bildschirmfüllende Ansicht kann somit aus mehreren Unteransichten zusammengesetzt sein, die zusammengenommen ein Gesamtbild der Applikation vermitteln. Android unterscheidet hierbei zwischen zwei Arten von Elementen: Views zum Darstellen von Daten und Bedienelementen sowie ViewGroups als Container für diese. Eine View kontrolliert hierbei laut [9] einen rechteckigen Bereich innerhalb des Fensters einer Activity und kann auf Benutzereingaben reagieren, wie beispielsweise ein Button-Element.

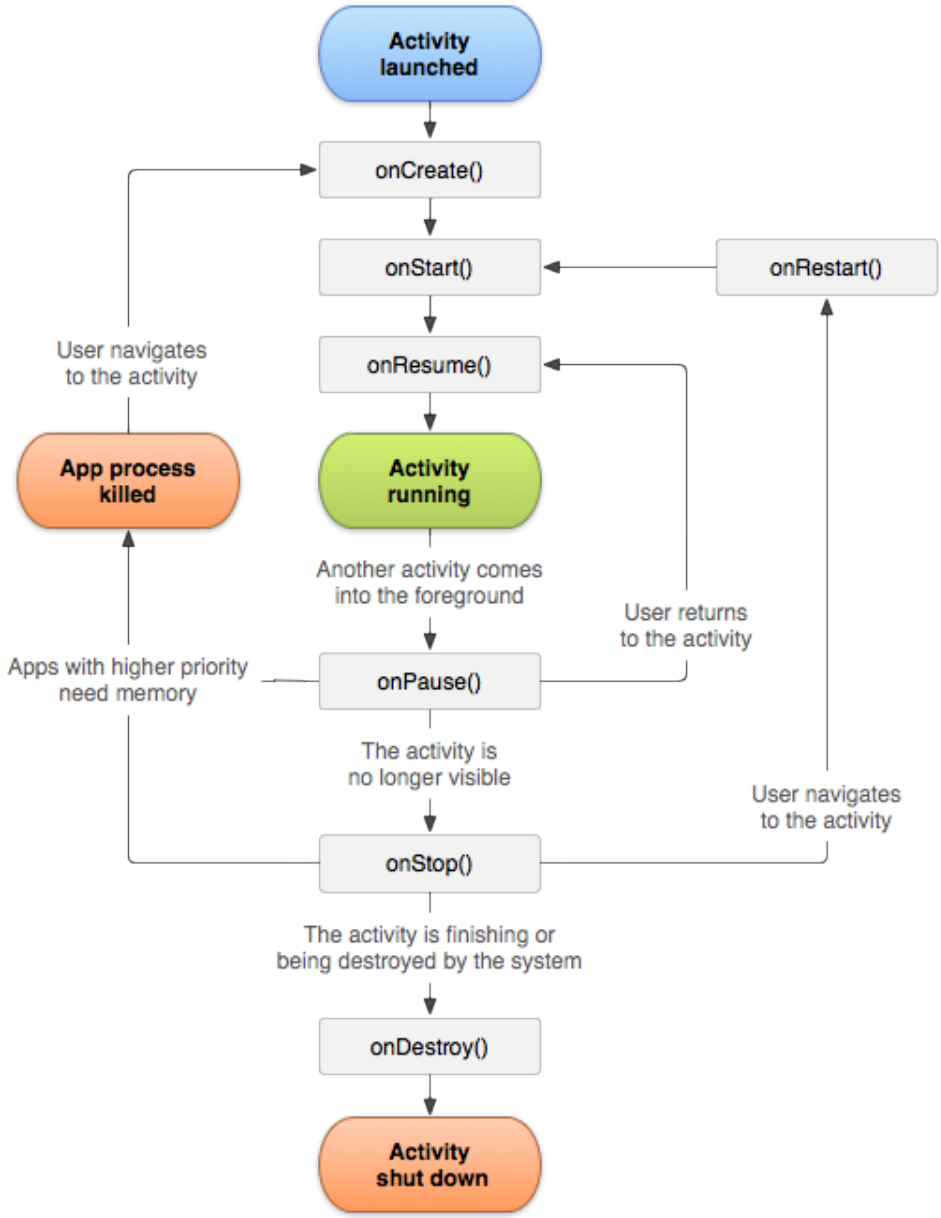


Abbildung 2.5: Der Lebenszyklus einer Android-Activity anhand der aufgerufenen Funktionen. [a]

[a] `http://developer.android.com/images/activity_lifecycle.png`

Die Sichten-Hierarchie ist in Android als Baum aufgebaut, bei dem jedes Unterelement genau einen Vaterknoten besitzt. Als oberstes Element dient meistens eine Layout-Klasse, die View-Elemente nach einem vorgegebenen Schema anordnet. Beispiele sind das LinearLayout, welches Elemente horizontal oder vertikal hintereinander anordnet, sowie das TableLayout, das Elemente tabellarisch in Zeilen und Spalten darstellt. Ein solches Layout dient in den meisten Fällen als Wurzel-Element der Sichten-Hierarchie einer Activity und wird mit *setConentView()* bei der Initialisierung gesetzt.

Views und Layouts können auch als XML-Dateien definiert und hinterlegt werden, welche zur Laufzeit geladen werden können. Dies dient vor allem der Trennung von Aussehen und Code. Zur Anordnung der einzelnen View-Elemente können die folgenden Layout-Klassen verwendet werden [11]:

LinearLayout Ordnet alle Elemente horizontal oder vertikal hintereinander an. Gut geeignet, um Listen von Elementen zu realisieren, beispielsweise für die Einstellungsansicht. Den Elementen können Gewichte zugeordnet werden, die ihre prozentuale Größe innerhalb des Layouts festlegen.

RelativeLayout Ordnet Elemente relativ zueinander an.

TableLayout Ordnet Elemente in Zeilen und Spalten an. Entspricht der Verwendung einer Tabelle.

Darüber hinaus existieren spezielle Views zum Gruppieren von Elementen, wie beispielsweise die *HorizontalScrollView*, die es ermöglicht, beliebig viele Elemente in einer scrollbaren Liste darzustellen.

2.4.4 Zeichnen unter Android

Android verwendet zum Zeichnen das Canvas-Konzept, welches eine leere Zeichenfläche zum Zeichnen von Objekten bereitstellt. Hierfür werden die folgenden vier Grundkomponenten benötigt [12]:

- Eine Bitmap zum Speichern der Pixel
- Eine Zeichenfläche (Canvas), um Draw Calls abzufangen
- Eine zu zeichnende Grundform (Drawing Primitive)
- Ein Pinsel (Paint), um die Art der Zeichnung festzulegen

Das Canvas dient hierbei als Schnittstelle zur eigentlichen Zeichenfläche für die umgebende Anwendung, der Bitmap. Dieses wird im Anzeigefenster platziert (Window). Das Zeichnen erfolgt durch Aufrufe der bereitgestellten draw()-Methoden. Das Pinselobjekt legt Farbe und Stil der Zeichnung der Grundform fest, beispielsweise ein Rechteck, ein Pfad, Text oder eine weitere Bitmap.

Um die Zeichenobjekte (Drawables) auf die Zeichenfläche zu bringen, gibt es in Android die folgenden zwei Ansätze [10]:

Indirektes Zeichnen Zeichnen der Grafik oder Animation in ein View-Objekt. Der Zeichenprozess wird hierbei vom System und dessen View-Hierarchie übernommen.

Direktes Zeichnen Zeichnen der Grafik direkt in ein Canvas. Hierbei muss die `onDraw()`-Methode der jeweiligen Klasse selbstständig aufgerufen werden, mit dem Canvas als Übergabeparameter. Alternativ können weitere `draw()`-Methoden aufgerufen werden.

Indirektes Zeichnen eignet sich vor allem für statische Grafiken und vordefinierte Animationen. Direktes Zeichnen wird vor allem bei leistungsintensiven Applikationen, wie beispielsweise Spielen, notwendig, die sich regelmäßig neu zeichnen müssen. Dabei existieren wiederum zwei Ansätze:

- Innerhalb der selben UI-Activity durch den Aufruf von `Invalidate()` und der Behandlung des `onDraw()`-Callbacks.
- In einem separaten Thread über eine `SurfaceView()`. Zeichenaufrufe werden so schnell wie möglich entsprechend des Threads durchgeführt, es ist kein Aufruf von `Invalidate()` nötig.

Für den zu implementierenden Prototyp genügt die Verwendung des indirekten Zeichnens, da nur ein statischer Graph, sowie statische Details angezeigt werden müssen.

3 Zielsetzung

Das vorliegende Kapitel beschreibt die konkrete Aufgabenstellung der Diplomarbeit, wie sie durch das Institut vorgegeben ist. Beginnend mit der offiziellen Ausschreibung, wird die Aufgabe klar definiert und gegenüber ähnlichen Teilbereichen abgegrenzt. Ebenso werden die technischen Rahmenbedingungen für die Durchführung der Arbeit festgelegt, unter denen der Prototyp zu entwickeln ist. Am Ende des Kapitels sind schließlich alle funktionalen Anforderungen aus der Sicht des Benutzers als Use-Cases dargestellt.

3.1 Ausschreibung

Filter/Flow-Graphen sind eine Form zur Visualisierung von booleschen und prädikatenlogischen Ausdrücken, wie sie oft in Datenbankabfragesprachen verwendet werden. Hierbei werden Filter durch Knoten repräsentiert und Objektbezüge durch Kanten. Somit können auch unerfahrene Nutzer komplexe Ausdrücke erstellen und evaluieren [35].

In der Diplomarbeit „Visualisierung von Filter/Flow-Graphen auf mobilen Endgeräten“ geht es um die Darstellung dieser Graphen auf Smartphones mit kleinen Bildschirmen. Hierbei soll ein Filter/Flow-Graph untersucht und manipuliert werden können. Darüber hinaus soll die kollaborative Zusammenarbeit an solchen Graphen ermöglicht werden.

Es soll ein Konzept entwickelt werden, um Filter/Flow-Graphen auf kleinen, berührungsempfindlichen Bildschirmen passend darzustellen. Hierbei sollen die Inhalte und Einstellungen von Knoten geändert werden können. Zudem soll die Struktur des Graphen durch Einfüge-, Lösch- und Verknüpfungsoperationen änderbar sein. Die gewählten Interaktionen sollen vom Benutzer gut mit dem kleinen Display umgesetzt werden können.

Wichtig für die Entwicklung des Konzeptes ist die Orientierungsmöglichkeit des Benutzers beim Wechsel zwischen der Desktop-Visualisierung und dem Mobilgerät – diese darf bei einem raschen Wechsel nicht verloren gehen. Es müssen somit Aspekte beachtet werden, die einen visuellen Zusammenhang zwischen den Darstellungen herstellen und dem Benutzer die Arbeit an zwei Bildschirmen erleichtern.

Im Rahmen dieser Arbeit soll eine Recherche durchgeführt werden, um Erkenntnisse über relevante Themengebiete zu erhalten. Anschließend soll ein Konzept entwickelt und dokumentiert, sowie ein Prototyp des entwickelten Konzeptes erstellt werden. Am Ende der Arbeit steht eine Evaluierung des entwickelten Konzeptes mit Hilfe einer Benutzerstudie.

3.2 Abgrenzung

In der vorliegenden Diplomarbeit liegt der Fokus auf der Visualisierung von Filter/Flow-Graphen, das heißt explizit deren Darstellung auf kleinen Displays. Weitere Darstellungsarten von komplexen, booleschen Anfragen werden in diesem Zusammenhang nicht untersucht. Die Nutzung der Filter/Flow-Repräsentation ist fest vorgegeben. Dies wird auch dadurch bedingt, dass sich die Ergebnisse dieser Arbeit in das bestehende Projekt zur Darstellung von Filter/Flow-Graphen auf dem Desktop integrieren soll.

3.3 Technische Einschränkungen

Eine der relevanten Einschränkungen betrifft die verfügbare Anzeigefläche der Zielplattform. Bei der Auswahl des Entwicklungsgerätes wurde zunächst auf die benötigte Bildschirmgröße geachtet. Da die Bildschirmgrößen von Smartphones zum Stand der Arbeit mit Größen von drei bis fünf Zoll sehr uneinheitlich sind, orientiert sich die Diplomarbeit am derzeitigen Nutzertrend. Dieser geht immer mehr Richtung größerer Displays - Stand 2012 würden sich 90% der existierenden Smartphone-Benutzer bei Neuerwerb eines Gerätes für ein größeres Display entscheiden [5]. Insgesamt bevorzugen Smartphone-Nutzer Geräte mit Bildschirmgrößen zwischen 4 und 4,5 Zoll aufgrund zunehmender Internetnutzung und der wachsenden Anzahl an multimedialen Angeboten.

Das entwickelte Konzept soll auf einem Mobilgerät mit Android-Betriebssystem als Prototyp implementiert werden, siehe Kapitel 6.1. Die Entscheidung für das Entwicklungsgerät fiel somit auf ein *Galaxy Nexus*, da es über ein 4,65 Zoll großes Display verfügt. Die verwendete Bildschirmgröße liegt im Trend aktueller Smartphone-Modelle und kann somit als Referenzgerät verwendet werden. Abbildung 3.1 zeigt das *Galaxy Nexus* aus allen drei Betrachtungswinkeln.

Das *Galaxy Nexus* wurde von Google entwickelt, die Hardware von Samsung gefertigt. Somit verfügt es über ein reines Android-Betriebssystem. Dies bietet eine Reihe von Vorteilen. Zum Einen bietet Google als einziger Software-Hersteller fortlaufende Updates für das installierte Betriebssystem seiner Geräte an, zum Stand der Arbeit Android Version 4.1, „Jelly Bean“. Zum Anderen wurde das System nicht durch herstellerspezifische Oberflächen und Applikationen verfälscht. Aus diesen Gründen eignet sich das Galaxy Nexus besonders als Entwicklergerät für die Android-Plattform.

Das Galaxy Nexus verfügt über ein LED-Display mit einer Auflösung von 1280 x 720 Pixel, das auch kleine Schriftgrößen scharf darstellen kann. Dies ist für die saubere Darstellung des Visualisierungskonzepts notwendig und ermöglicht es den Bildschirmplatz optimal auszunutzen. Der Prozessor verfügt mit zwei Kernen und 1,2 GHz über ausreichend Leistung, um die Anforderungen umzusetzen. Der interne Speicher ist mit 16GB ebenso groß genug, um selbst aufwendige Grafiken ablegen zu können. Tabelle 9.1 im Anhang zeigt alle Hardwarespezifikationen des Galaxy Nexus.

Abbildung 3.1: Das Google Galaxy Nexus. [a]

[a] http://www.google.de/nexus/a-images/phone-galaxy-gallery-family.png

3.4 Anforderungen

Die Aufgabe umfasst die Entwicklung eines eigenständigen Konzeptes zur Betrachtung und Bearbeitung von Filter/Flow-Graphen. Hierzu gehören die folgenden Anforderungen, die durch das Konzept erfüllt werden müssen:

Auswahl beliebiger Knoten Der Nutzer soll in der Lage sein, jeden beliebigen Knoten als Startpunkt wählen zu können, um ihn und dessen unmittelbare Umgebung anzuzeigen oder zu editieren.

Ansicht ausgewählter Knoten Ausgewählte Knoten sollen inklusive ihrer Umgebung darstellbar sein. Dazu gehören sämtliche zu ihm gehörigen Attribute, sowie Kanten zu abhängigen Knoten.

Editieren von Attributen Der Nutzer soll in der Lage sein, Attribute ausgewählter Knoten zu editieren.

Hinzufügen und Löschen von Knoten Der Nutzer soll ausgewählte Knoten löschen und neue hinzufügen können.

Verknüpfen von Knoten Der Nutzer soll Relationen zwischen Knoten, in Form von Kanten, hinzufügen und löschen können.

Übersichtlichkeit Beim Wechsel zwischen Mobilgerät und Desktop darf die Übersicht nicht verloren gehen. Der Benutzer soll stets wissen, welchen Teil des Graphen er editiert.

Leistungsfähigkeit Die Ressourcen der Mobilplattform sollen möglichst effizient genutzt werden. Größere Rechenaufgaben können an den Server delegiert werden. Da die Implementierung nur prototypisch erfolgt, muss diese jedoch nur mit den gewählten Testgraphen ausreichend bedienbar sein.

3.5 Lösungsansatz

Um die gesetzten Ziele zu erreichen werden zunächst themenverwandte Arbeiten zu den Themen Visualisierung von Graphen sowie Darstellungsarten auf mobilen Endgeräten betrachtet. Hierbei liegt die Gewichtung vor allem auf einer übersichtlichen Darstellung und einer guten Bedienbarkeit von Konzepten auf kleinen Bildschirmen. Basierend auf diesen Arbeiten und den in diesem Kapitel definierten Anforderungen wird ein eigenes Konzept entwickelt, dass sich dazu eignet, die Zielsetzungen zu erreichen. Die einzelnen Anforderungen werden anschließend mit Hilfe dieses Konzeptes formuliert, um die benötigten Interaktionen und erwarteten Ergebnisse zu definieren.

Bei der Entwicklung werden die Grundlagen aus Kapitel 2 mit eingebracht – das Konzept soll in der Lage sein, die Datenstrukturen aus Kapitel 2.1.4 darstellen zu können und muss gleichzeitig mit den gegebenen technischen Mitteln aus Kapitel 2.4.1 umgesetzt

werden können. Diese spielen bei der Implementierung in Kapitel 6 eine vorrangige Rolle. Das Konzept wird an die möglichen Darstellungsarten des Betriebssystems Android angepasst, wobei besonders auf die Eingabemöglichkeiten durch interaktive Touch-Elemente geachtet wird. Beim Entwurf der Architektur des Prototyps soll die Architektur der Referenzimplementierung untersucht werden, um auf ihr aufzubauen. Diese wird in Unterabschnitt 4.2.2 des kommenden Kapitels beschrieben.

4 Themenbezogene Arbeiten/Stand der Forschung

4.1 Bedienkonzepte

Im Folgenden werden grundlegende Bedienkonzepte bei der Visualisierung beschrieben und hinsichtlich der Aufgabenstellung bewertet. Jedes Bedienkonzept kann sich hierbei verschiedener Layouts zur effizienten Darstellung auf der gewählten Zielplattform bedienen.

4.1.1 Overview first, Zoom, Details on Demand

Der klassische Ansatz „Overview first, zoom and filter, then details-on-demand“ zur Visualisierung von Informationen wird von Shneiderman als *Visual Information Seeking Mantra* hervorgehoben [31]. Das Konzept sieht vor, Informationen zunächst übersichtlich zu präsentieren, mit der Möglichkeit, Teilbereiche zu vergrößern und Details sichtbar zu machen. Somit werden Strukturen und Abhängigkeiten auf Anhieb sichtbar, feine Details können bei Bedarf hinzugeschaltet werden. Das Konzept wird von vielen älteren Ansätzen verwendet und eignet sich für Anwendungsfälle, bei denen große Anzeigen zum Einsatz kommen. Die darstellbare Graphen- beziehungsweise Informationsgröße unterliegt jedoch technischen Einschränkungen, da für diesen Ansatz grundsätzlich der komplette Graph zur Berechnung der Übersicht verfügbar sein muss. Des Weiteren werden ab einer bestimmten Größe zusätzliche Techniken notwendig, um die Übersicht zu behalten, wie beispielsweise Clustering.

4.1.2 Search, Show context, Expand on Demand

Ein neues Konzept behandelt die Darstellung von Teilgraphen über eine Degree-of-Interest-Funktion [15]. Es wurde für die Betrachtung großer Graphen über eine Internetverbindung oder ähnliches konzipiert, bei denen Ressourceneinschränkungen eine Rolle spielen. Die Autoren gehen davon aus, dass in vielen Anwendungsfällen von Graphvisualisierung der komplette Graph nicht zur Verfügung steht oder berechnet werden kann. Dies kann von unterschiedlichen Benutzerrechten, sowie technischen Einschränkungen herrühren, wie es bei mobilen Endgeräten der Fall ist. Deshalb verfolgen die Autoren den Ansatz, von einem ausgesuchten Punkt in der Visualisierung zu starten und dem Nutzer nur relevante

Informationen des Kontext anzuzeigen. Auf Wunsch kann der Nutzer dann die angezeigte Informationsmenge erhöhen und weitere Knoten in der Umgebung aufdecken.

Der Ansatz geht davon aus, dass zunächst ein entsprechender Suchalgorithmus notwendig ist, um einen Startpunkt zu finden. Dieses kann beispielsweise über eine Textliste oder einer Suchanfrage an den Server geschehen. Anschließend muss über eine definierte Degree-Of-Interest-Funktion der anzuzeigende Kontext des fokussierten Suchknotens berechnet werden. Diese Funktion muss definiert werden und ergibt sich in bestimmten Anwendungsfällen aus natürlichen Eigenschaften der Datenmodelle. Über diesen angezeigten Kontext hinaus werden interessante Expansionsmöglichkeiten als farbig markierte Kanten dargestellt, um dies dem Benutzer anzuzeigen.

Im referenzierten Artikel wird eine Client-Server-Umsetzung getestet, die für mobile Clients mit beschränkten Ressourcen ausgelegt wurde. Somit eignet sich das Konzept für Mobilgeräte und kann in einer abgewandelten Form für die Umsetzung der Aufgabenstellung herangezogen werden. Jedoch muss zunächst die Degree-Of-Interest-Funktion auf den verwendeten Anfragekonstrukten definiert werden. Eine Möglichkeit wäre, diese Funktion aus vergangenen Nutzeraktionen abzuleiten, einer Art Historie expandierter Knoten.

4.2 Filter/Flow

Das folgende Kapitel beschreibt vorhandene Implementierungen von Filter/Flow-Graphen und deren Einfluss auf das zu entwickelnde Konzept.

4.2.1 Prototyp von Shneiderman

Der originale Prototyp einer Filter/Flow-Graphen-Darstellung geht auf Degi Young und Ben Shneiderman zurück [35]. In Kapitel 2.1 beschreiben die Autoren den Aufbau Ihrer Applikation. So ist das Interface in drei Teilbereiche unterteilt, wie in den Abbildungen 4.1 und 4.2 zu sehen ist. An einem Bildschirmrand findet sich die Datenbankliste mit den Datenquellen, auf der gegenüberliegenden Seite die Ergebnismenge. Diese bestehen aus solchen Datentupeln, die den gesetzten Filtern der Anfrage genügen. Die Anfrage selbst wird als Graph im mittleren Bildschirmabschnitt dargestellt.

Am oberen Rand werden Attributnamen, wie beispielsweise „Manager“ oder „Salary“, als Buttons dargestellt. Diese lassen sich anklicken, wodurch das entsprechende Attributmenü im Anfragenfeld erscheint. Das Attributmenü lässt sich an beliebige Stellen im Anfragefeld verschieben, um eine Anfrage zu modellieren. Durch Klicken auf das Attribut lässt sich der Wert, dem das Attribut genügen muss, festlegen. Die logische Verbindung der Attribute wird durch die Anordnung an einem virtuellen Gitter gegeben. Horizontal angeordnete Attribute werden mit einem logischen „Und“ verknüpft, vertikal angeordnete Elemente mit einem logischen „Oder“. Ein logisches „Oder“ kann hierbei

auch innerhalb eines Attribut-Elements für Werte erfolgen, indem mehrere dieser Werte ausgewählt werden. Im Gegensatz zum verwendeten Konzept, werden in Shneidermans Prototyp keine Kanten zum Verbinden der Knoten gezogen.

Das logische „Nicht“, die Invertierung einer Menge, wurde im Prototyp durch einen „Nicht“-Operator unterstützt [29]. Dieser sollte die Auswahl innerhalb eines Filters invertieren, indem er alle ausgewählten Werte abwählt und alle restlichen Elemente auswählt. Zudem wurde das Clustering von Filtern unterstützt, um größere Queries komplett auf dem Bildschirm darstellen zu können. Cluster sollten hierbei für den späteren Einsatz gespeichert werden können.

Am unteren Rand befindet sich zudem eine Aktionsleiste. Durch Klicken auf „Flow“ können Nutzer die logische Repräsentation der Query betrachten, die sie soeben über die Anordnung der Attribute erstellt haben. Diese wird durch die Analogie zu Wasser, welches durch die erstellten Filter fließt, dargestellt. Aus dem in [35] beschriebenen Konzept geht nicht genau hervor, ob beim Anlegen von Attributen bereits eine Darstellung von Kanten zwischen diesen erfolgt, um dem Benutzer die Zuordnung zu erleichtern. Es wird vermutet, dass dies nicht der Fall war und sich somit negativ auf das Verständnis des erstellten Graphen ausgewirkt haben könnte.

In einer weiteren Skizze von Shneiderman [30] wird der Datenfluss vertikal dargestellt. Vertikale Anordnungen stellen „Und“-Verknüfungen, horizontale „Oder“-Verknüpfungen dar. Shneiderman bezieht sich hier auf dynamische Anfragen, bei denen sich die Ergebnismenge dynamisch an Änderungen der Anfrage anpasst. Der Nutzer sieht somit sofort, welchen Einfluss seine Änderungen auf das Ergebnis haben. Des Weiteren wird die Ergebnismenge anhand einer Karte visualisiert, um dem Benutzer das Verständnis zu erleichtern. Abbildung 4.3 zeigt die Skizze, die für Filter/Flow-Graphen entworfen wurde.

„Und“-Operator - Schnittmengenbildung

Abbildung 4.1 zeigt die Darstellung eines logischen „Und“, das zur Bildung einer Schnittmenge führt. Hierfür sind mindestens zwei Bedingungen notwendig, in diesem Fall *Location = California* und *Salary = 50,000*. Diese müssen sich im Arbeitsbereich der Filter/Flow-Darstellung in derselben Zeile, jedoch in verschiedenen Spalten befinden. Somit erfolgt eine Verkettung, in der die Ergebnistupel des ersten Filters als Eingang für den darauf folgenden Filter dienen und nur die Ergebnisse übrig bleiben, die beiden Bedingungen genügen. Der dargestellte Informationsfluss passt seine Breite an die Menge der Ergebnistupel an, die hinter jedem Filter übrig bleiben, relativ zur Gesamtmenge, die in den Filter geflossen ist.

„Oder“-Operator - Vereinigungsoperator

Abbildung 4.1 zeigt die Darstellung des logischen „Oder“-Operators. Es sind ebenfalls zwei oder mehr Bedingungen notwendig, die nun in einer Spalte angeordnet werden

müssen, aber in unterschiedlichen Zeilen. Die Ergebnismengen werden nun vereinigt; es gelangen alle Tupel durch die Filter, die mindestens einer Bedingung genügen. Im vorliegenden Fall sind dies alle Arbeiter mit Manager *Deji* oder der Arbeitsbezeichnung *Driver*.

„Oder"-Verknüpfungen können jedoch auch innerhalb eines Filters, durch die Auswahl mehrerer Attribute dem der Filter genügen muss, dargestellt werden [29].

4.2.2 Filter/Flow-Referenzimplementierung

Bei der Referenzimplementierung handelt es sich um eine Desktop-Applikation, die zur Erstellung und Bearbeitung von erweiterten Filter/Flow-Graphen dient und das Konzept aus Kapitel 2.1.4 implementiert. Die Applikation wurde vom Institut für Visualisierung und Interaktive Systeme implementiert; der Quellcode steht somit zur Verfügung. Der Quellcode der Applikation wurde in C# geschrieben, das Programm basiert auf dem .NET Framework und nutzt *WPF* zur Darstellung des Graphen.

Bei der Implementierung können Knoten auf der Oberfläche frei angeordnet werden. Emitter können bei Bedarf hinzugefügt oder gelöscht werden. Eine neue Kante (Flow) kann durch Klicken und Ziehen von Emitter zu Rezeptor und umgekehrt erstellt werden. Im Gegensatz zum Filter/Flow-Prototyp von Shneiderman aus Kapitel 4.2.1, spielt die horizontale oder vertikale Position der Filter keine Rolle für die semantische Auswertung des Filterausdrucks. Hierzu werden lediglich die Kanten ausgewertet, durch die die Knoten verbunden sind. Sequentielle Verbindungen bilden ein logisches „Und", parallele Verbindungen ein logisches „Oder".

Die Projektstruktur des Quellcodes dient als Ausgangsbasis für die Implementierung des Prototyps. Ebenso dient das Datenmodell als Basis für den Prototyp. Später ist eine Kommunikation zwischen der vorhandenen Implementierung und dem Prototyp vorgesehen, weshalb eine gemeinsame Web-Service-Schnittstelle vorgesehen ist.

Abbildung 4.4 zeigt einen Ausschnitt aus der Referenzimplementierung. Man sieht die Umsetzung der erweiterten Knotentypen aus Kapitel 2.1.4. Es sind beispielsweise bedingte Filter, binäre Operatoren, sowie Between-Filter erkennbar. Der Prototyp orientiert sich bei der graphischen Implementierung an der Umsetzung der Desktop-Version. Dies äußert sich beispielsweise bei der Nutzung der gleichen Farbwerte für Knotentypen, Konnektoren und Kanten.

4.3 Minimap

Nokia beschreibt ein Verfahren namens „Minimap", um die Probleme aus Kapitel 2.3.3 für das Betrachten von Webseiten auf kleinen Bildschirmen zu lösen [27]. Die Autoren haben sich mit den naiven Ansätzen zur Darstellung von Webseiten auseinandergesetzt. Ihre Lösung besteht aus einem Kompromiss zwischen Strukturerhalt der Originalseite

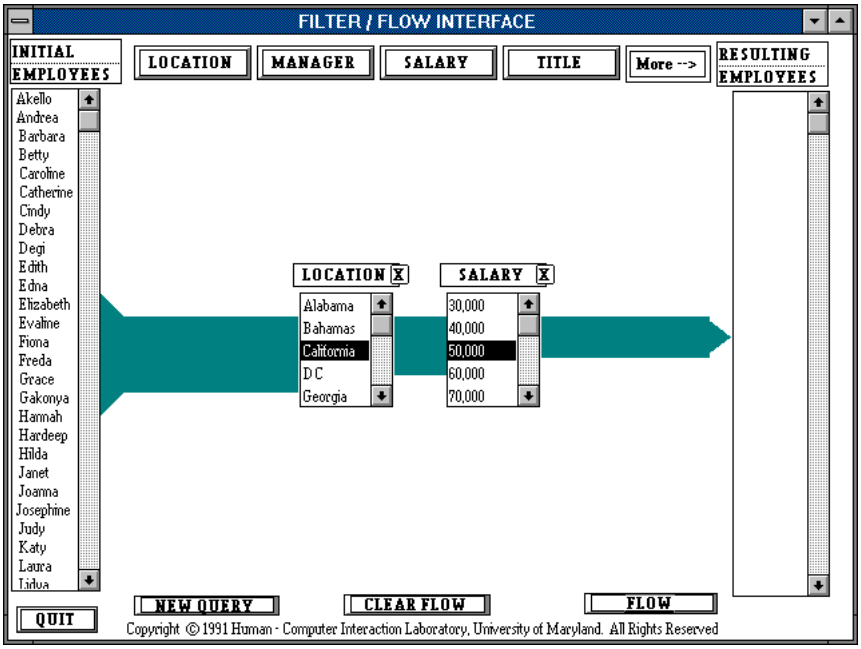


Abbildung 4.1: Filter/Flow-Repräsentation des booleschen „Und“-Operators, innerhalb der Anfrage *(Location = California) AND (Salary = 50,000)*

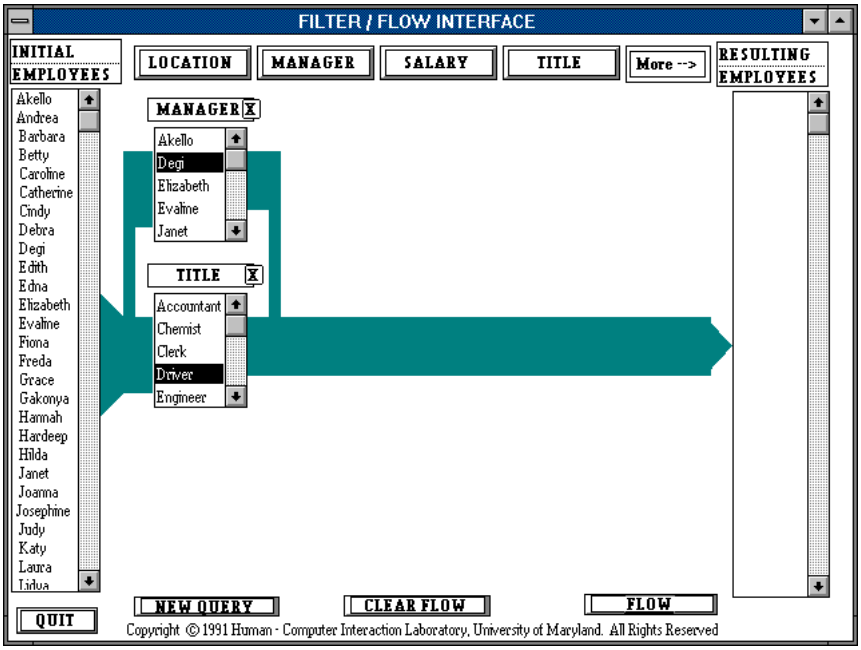


Abbildung 4.2: Filter/Flow-Repräsentation des booleschen „Oder“-Operators, innerhalb der Abfrage *(Manager = Deji) OR (Title = Driver)*

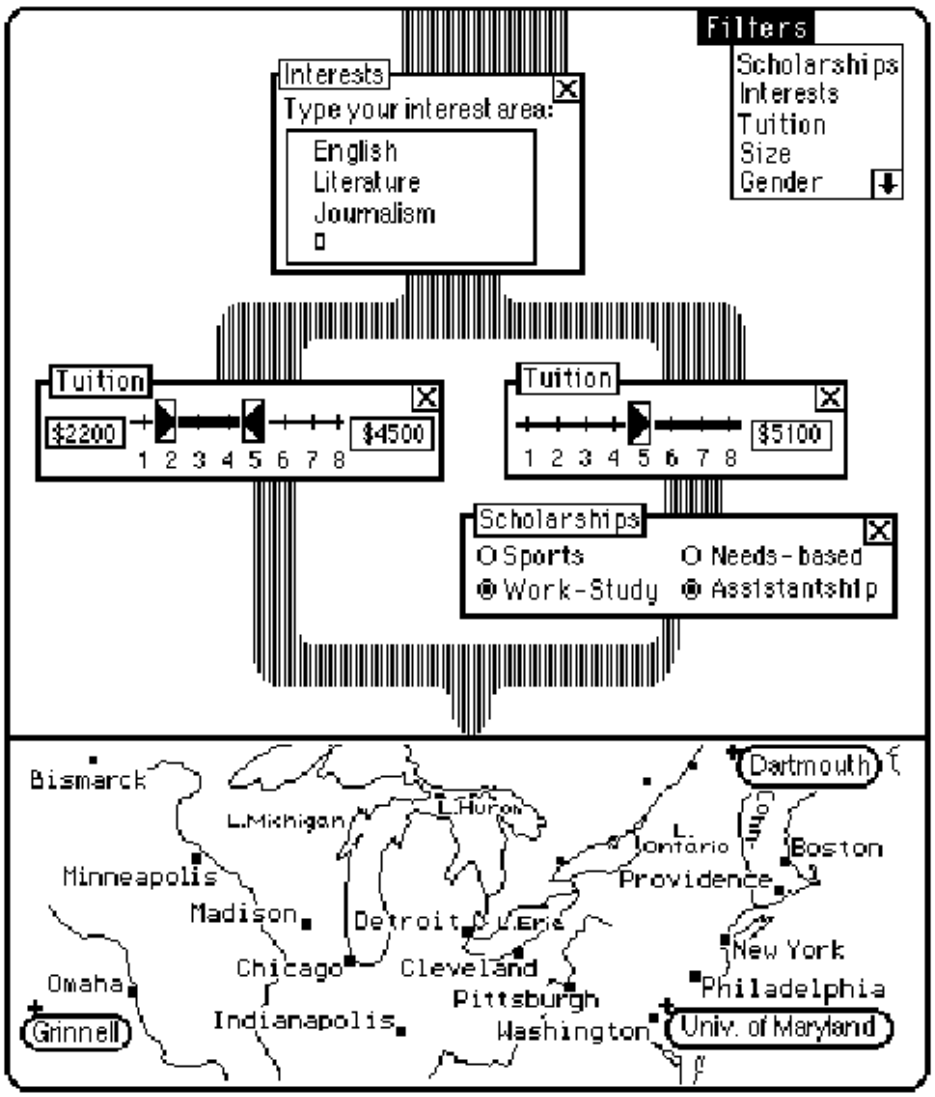


Abbildung 4.3: Skizze einer booleschen Filter/Flow-Anfrage, inklusive einer Kartenanzeige zur Darstellung der Ergebnismenge. Die Abfrage lautete: *(Interests = English or literature or journalism) and ((tuition greater or equal to $2,200 or less than or equal to $4,500) or ((tuition greater than or equal to $5,100) and (scholarships are available by work-Study or assistantship)).*

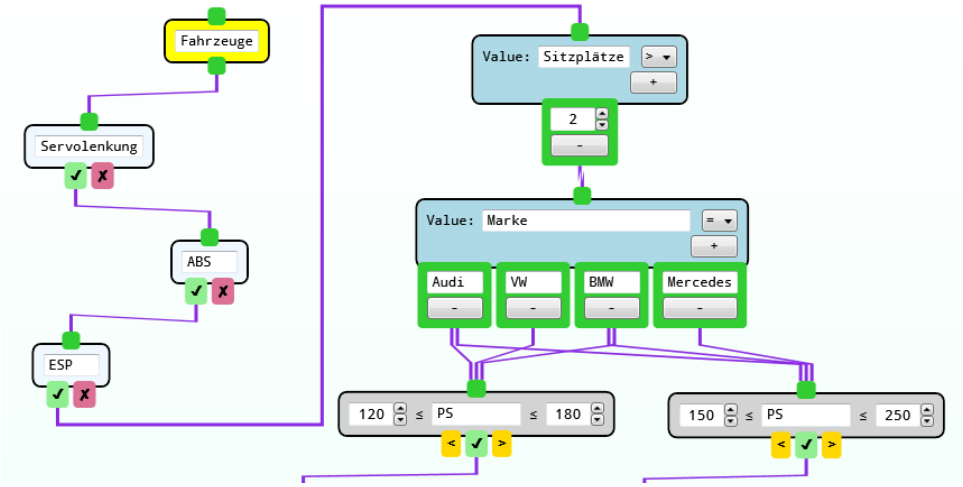


Abbildung 4.4: Ausschnitt aus der Referenzimplementierung für Filter/Flow-Graphen. Es sind Umsetzungen der Erweiterungskonstrukte aus Kapitel 2.1.4 sichtbar.

und Lesbarkeit des in Kapitel 2.3.3 beschriebenen „Narrow Layouts“. Dies wird erreicht, indem Text auf eine lesbare Größe gebracht wird, abhängig von der Bildschirmgröße und ungenutzte Bereiche zwischen Inhaltselementen entfernt werden. Die relative Struktur und damit die Navigationsmöglichkeit sollen erhalten bleiben. Gleichzeitig wird Text, der nicht in den Bereich des Viewports, also die Breite des Bildschirms passt, entsprechend umgebrochen. Bilder werden im Gegensatz zum „Narrow Layout“ nicht auf die maximale Breite des Viewports hinabskaliert, sondern lediglich auf bis zu 50 Prozent ihrer Originalgröße verkleinert.

Der vorgestellte Algorithmus arbeitet nach den folgenden Schritten:

1. Die Breiten- und Höhenbeschränkungen des Viewports werden mit einem Skalierungsfaktor multipliziert
2. Textmetriken, wie Breite und Höhe von Strings, werden während des Formatierungsschrittes mit dem Skalierungsfaktor multipliziert.
3. Falls die Textbreite eines Paragraphen breiter ist als die Breite des Viewports, so wird der Text umgebrochen, um in den Viewport zu passen.
4. Während des Zeichnens werden alle Koordinaten und Größen, die im Formatierungsschritt berechnet worden sind, durch den Skalierungsfaktor geteilt, und für die Darstellung auf der Bildschirmausgabe verwendet. Textinhalte werden hierbei wieder auf ihre Ursprungsgröße zurückgesetzt.

Zusätzlich zu dieser Darstellungsanpassung wird eine Minimap eingeblendet, von der sich der Name des Algorithmus ableitet. Diese wird als transparentes Overlay in einer Bildschirmecke über dem derzeitigen Viewport eingeblendet und zeigt mittels eines roten Rechtecks die derzeitige Position innerhalb der dargestellten Webseite. Somit kann der Anwender sofort sehen, wo er sich befindet, und wo er hinnavigiert. Zusätzlich werden Bereiche, in denen der Anwender noch nicht gewesen ist, halb-transparent in gelb eingefärbt. Somit kann er einfacher prüfen, welche Bereiche des Inhaltes er bereits besucht hat.

Das Konzept wurde zunächst in einer Nutzerstudie untersucht, um weitere Verbesserungen einbringen zu können. Hierbei stellte sich heraus, dass die Nutzer durch die ständige Einblendung der Minimap teilweise gestört waren, ihren Nutzen aber dennoch schätzten. Um den Anwender nicht zu irritieren, wurde die Minimap nachfolgend nur in Situationen eingeblendet, in denen der Benutzer längere Strecken durch Scrollen zurücklegt. Kurze Textabschnitte können somit ungestört gelesen werden, da die Minimap bei einzelnen Scroll-Schritten nicht eingeblendet wird.

Die Entwickler des Minimap-Konzepts haben eine Langzeit-Studie durchgeführt, bei der das Minimap-Konzept gegen das Narrow-Layout-Konzept antreten musste. Hierbei wurden zwanzig Probanden über sechzehn Tage lang mit beiden Browsertypen (Minimap und Narrow-Layout) konfrontiert [27] – 90% der Probanden bevorzugten nach dieser Evaluierung den Minimap-Browser, was eine deutliche Tendenz in Richtung des entwickelten Konzepts erkennen lässt. Der Hauptgrund besteht darin, dass das vertraute Layout der Originalwebseite bei der Benutzung des Minimap-Konzepts nicht verloren geht [27].

Es konnte gezeigt werden, dass das Betrachten großer Bilder und Tabellen bei der Minimap-Variante deutlich besser abschnitt. Dies ist damit zu erklären, dass Bilder nicht um mehr als die Hälfte verkleinert werden und Details somit besser erkennbar bleiben. Zudem kann ein Bild bei der Minimap-Variante nach wie vor gezoomt werden, was beim Narrow-Layout nicht vorgesehen ist. Tabellen werden bestmöglich erhalten und nicht innerhalb einer Zeile umgebrochen, was das Verständnis, wie aufgezeigt wurde, deutlich erschwert.

Der Wechsel vom Narrow-Layout zur Originalansicht zum Lösen mancher Aufgaben wurde nur von einem Teil der Probanden nachvollzogen, da diese Funktion nicht explizit ausgeschrieben ist. Sichtwechsel sollten somit für den Benutzer offensichtlich sichtbar und zugänglich sein. Ebenso interessant war das Ergebnis beim Lesen langer Texte. Diese schnitten auf beiden Browsern gleich gut ab, obwohl bei der Minimap-Variante Textblöcke auf die Breite des Viewports unterteilt sind.

Die Ergebnisse der Evaluierung zeigen, dass eine Zerstörung der inhaltlichen Struktur das Verständnis erschweren. Ein naiver Ansatz zur Komprimierung ist somit nicht empfehlenswert. Des weiteren trug die Anzeige der Minimap stark zur Übersicht und damit zur Navigation bei. Aus diesen Gründen soll für die Darstellung von Filter/Flow-Graphen, bei denen die Struktur mindestens ebenso wichtig zum Verständnis ist, ein Verfahren ähnlich dem Minimap-Ansatz gefunden werden.

4.4 Referenzapplikationen

Unter Referenzapplikationen sind Applikationen zu verstehen, die sich mit einem ähnlichen Thema wie die Diplomarbeit befassen. Hierzu zählen jene Applikationen für mobile Betriebssysteme, die sich mit der Darstellung von Graphen beschäftigen. Somit fallen hierunter Apps der drei größten Plattformen Android, iOS und Phone 7. Da dem Diplomanden nur Geräte der Android-Plattform zur Verfügung stehen, beschränkt sich die Suche somit auf diesen Teilbereich.

Es soll untersucht werden, wie die aufgeführten Applikationen mit der Darstellung der Graphen umgehen und wie deren Bedienung funktioniert. Es sollen zudem Vor- und Nachteile der jeweiligen Applikation festgehalten werden, um diese in die Entwicklung des eigenen Konzeptes miteinzubeziehen.

4.4.1 MindJet

MindJet dient der Erstellung von Mindmaps auf Desktopcomputern, sowie auf Tablets und Smartphones [21]. Der Benutzer kann einen Graphen aus einfachen Textknoten anlegen, die wiederum mehrere Unterknoten besitzen können. Die Teilbäume die sich hieraus ergeben kann der Benutzer mit einem Klick ein- oder ausklappen, um die Übersicht zu erhöhen. Die Funktion dient somit einem Details-On-Demand-Konzept.

Mindjet für Android implementiert eine Pan-And-Zoom-Oberfläche für die Darstellung des Graphen. Der Nutzer kann bis zu einem vorgegebenen Grad hinein und hinaus zoomen. Leider sieht er dabei nicht, auf welcher Zoomebene er sich derzeit befindet. Funktionen zum Bearbeiten des Graphen werden am linken Bildschirmrand in einer Menüleiste dargestellt. Über eine weitere Leiste am unteren Ende kann der Benutzer die Funktionen der Menüleiste durchschalten. Menüfunktionen werden hierbei auf den gerade ausgewählten Knoten angewandt. Die Anordnung der Elemente ist übersichtlich, der Benutzer muss jedoch bei verschiedenen Aktionen durch die Menüleiste scrollen und die gewünschte Funktion suchen. Zudem nutzt die Menüleiste einen Teil des Bildschirmbereiches der Graphoberfläche.

Das Einfügen von Knoten wird über die Menüleiste ausgeführt. Hierbei wird ein neuer Unterknoten am derzeit ausgewählten Knoten hinzugefügt. Um zwei Knoten zu verbinden, muss aus der Menüleiste das entsprechende Werkzeug ausgewählt werden. Der Nutzer muss nun den Zielknoten wählen, der ausgewählte Knoten dient als Startknoten für die Interaktion.

Vorteile

Übersichtliche Graphstruktur beim Herauszoomen

Menüleiste für komplexere Interaktionen

Nachteile

Umständliche Navigation in großen Graphen

Texte nicht mehr lesbar ab gewissem Zoom-Level

Knoten zum Ausklappen von Teilbäumen klein und schwer zu bedienen

Menüleiste nimmt beim Bearbeiten einen Teil des Bildschirmbereichs ein

MindJet bietet gute Ansätze zur Darstellung von Graphen. Filter/Flow-Graphen stellen jedoch komplexe Filterausdrücke dar und bieten somit umfangreichere Knoteninhalte. Es muss somit ein Konzept gefunden werden, wie die Knoteninhalte bearbeitet werden können. Die Strukturdarstellung des Graphen ist gut, man kann jedoch ab einem gewissen Zoom-Level keine Knoten-Beschriftungen mehr lesen. Das Verbinden von Knoten funktioniert für einfache Textknoten ohne Konnektoren gut, muss aber bei Filter/Flow-Graphen durch eine genauere Variante ersetzt werden.

4.5 Abgrenzung von den vorhandenen Arbeiten

Das zu entwickelnde Konzept soll eine Übersicht über einen kompletten Filter/Flow-Graphen liefern und gleichzeitig in der Lage sein, Details jedes einzelnen Knotens editierbar darzustellen. Es wurde aufgezeigt, dass der Erhalt der Graphenstruktur für die Orientierung des Benutzers unverzichtbar ist. Anhand der Referenzapplikation MindJet aus Kapitel 4.4 ist ersichtlich, dass eine Graphstruktur mit abstrahierten Details gut auf dem kleinen Bildschirm eines Smartphones dargestellt werden kann. Allerdings ist die Bedienung kleiner Elemente über den Touchscreen, bei gleichzeitiger Navigationsmöglichkeit, schwierig.

Den Ansatz einer Minimap aus Kapitel 4.3 bietet hier gute Voraussetzungen. Es könnte eine Art Minimap verwendet werden, um den Gesamtgraphen darzustellen. Auf dieser Minimap des Graphen kann die Position des Viewports und dessen Ausschnitt durch ein Rechteck dargestellt werden. Dieses soll durch den Benutzer verschiebbar sein, um den Inhalt für den Viewport auswählen zu können.

Da es bei dem Minimap-Ansatz Probleme mit der ständigen Einblendung gab, soll der Inhalt des Viewports getrennt von der Ansicht einer Minimap dargestellt werden. Hierzu kann sich das Konzept bei dem von Shneiderman bekannten Ansatz aus Kapitel 4.1.1 bedienen, der Details und Übersicht voneinander trennt. Der Viewport soll stets den ausgewählten Ausschnitt im Detail darstellen, der Benutzer soll jederzeit in die jeweils andere Ansicht wechseln können. Eine Navigation innerhalb des Graphen soll in beiden Ansichten möglich sein.

Die Festlegung auf eine getrennte Darstellung der Minimap und Detailansicht erfolgte aus den folgenden Gründen:

- Benutzer waren in der ersten durchgeführten Studie durch das ständige Einblenden der Minimap bei jedem Scrollvorgang irritiert [27]. Eine getrennte Darstellung, verbunden durch eine Tab-Leiste soll dieses Problem beheben.
- Der Wechsel zwischen verschiedenen Ansichten ist ein gängiges Konzept von modernen Smartphone-Betriebssystemen, an die die Benutzer bereits gewöhnt sind (siehe Kapitel 2.3.2). Die Einblendung eines Overlays muss dagegen noch erforscht werden.
- Das Editieren von Elementen wurde im Minimap-Konzept nicht explizit vorgesehen und bei der Evaluierung nicht berücksichtigt. Dies stellt jedoch einen wichtigen Anwendungsfall für die Diplomarbeit dar. Es ist somit fraglich, ob das Konzept ohne Weiteres ein leichtes Editieren zulässt.
- Die Trennung erlaubt beiden Ansichten nahezu den kompletten Bildschirmbereich zu benutzen, abzüglich der Navigationsleiste. Somit kann die Minimap die Struktur von vergleichsweise großen Graphen darstellen. Gleichzeitig können in der Detailansicht mehr Details dargestellt werden, da nur ein Knoten auf einmal angezeigt wird.
- Ebenso erlaubt die Trennung eine Einbindung von an Mobilgeräte angepassten Bedienelementen in die Detailansicht. Diese müssen über eine bestimmte Mindestgröße verfügen, um sie per Touch-Eingabe bedienen zu können.

Ebenso wird der Komprimierungs-Ansatz für die Gesamtansicht übernommen. Hierbei werden nur die nötigsten Informationen, wie der Knotentyp, dargestellt, um den Knoten zu identifizieren. In der Detailansicht findet sich dann der komplette Inhalt des aktuellen Viewports, um diesen einsehen und editieren zu können.

5 Lösungskonzept

Dieses Kapitel beschreibt das eigens entwickelte Konzept zur Umsetzung der Aufgabenstellung und Lösung der in Kapitel 3 beschriebenen Probleme. Verwendete Ansätze wurden in Kapitel 4 bereits hergeleitet und als Lösungsansatz beschrieben. Dieser wird in den folgenden Unterkapiteln konkret ausformuliert.

5.1 Eigenes Konzept

Bei der Suche nach einem geeigneten Konzept wurden die Ansätze aus Kapitel 4.1 in Betracht gezogen. Diese konnten jedoch für sich allein nicht alle gestellten Anforderungen erfüllen. Es wurden somit vor allem Ideen aus Kapitel 4.3 übernommen, um die formulierten Ziele zu erfüllen.

Das entwickelte Konzept umfasst die Entwicklung einer reinen Übersichtsansicht, sowie einer Detailansicht zur Bearbeitung von Einträgen. Dies wird, unter anderem, durch die Möglichkeiten des verwendeten Betriebssystems Android bedingt, das eine Implementierung in einer einzigen Ansicht erschwert, siehe Kapitel 6.3. Das entwickelte Konzept vereinigt die positiven Aspekte der vorgestellten Ansätze. Dabei sollen die folgenden Eigenschaften enthalten sein:

Overview Der Benutzer ist in der Lage, die Struktur des ganzen Graphen zu überblicken und eine Stelle auszuwählen, die bearbeitet werden soll.

Details on Demand Der Benutzer kann einen gewählten Ausschnitt der Gesamtansicht in einer Detailansicht betrachten, um diesen zu untersuchen oder zu bearbeiten. Dies ist notwendig, um alle Knotendetails gut lesbar darzustellen und eine präzise Bedienung über den berührungsempfindlichen Bildschirm zu gewährleisten.

Show Context Dem Benutzer wird in der Detailansicht die unmittelbare Umgebung des betrachteten Knotens angezeigt. Hierzu zählen vor allem angeschlossene Kanten. Somit weiß der Benutzer stets, in welchem Verhältnis der Knoten zu seiner Umgebung steht.

Expand on Demand Der Benutzer kann einem Flow (den Kanten im Graphen) in der Detailansicht folgen, um vorausgehende und nachfolgende Filterknoten zu betrachten.

Hierbei ergeben sich die folgenden Herausforderungen:

Detailreduktion In der Übersichtsansicht muss ein Konzept zur Reduktion der Datenstrukturen auf ihre Struktur gefunden und implementiert werden. Zur kollaborativen Arbeit soll die Struktur der Übersicht mit der Struktur auf dem Hostgerät übereinstimmen.

Detailansicht Es muss eine Detailansicht entwickelt werden, die es erlaubt, alle geforderten Datenmanipulationen vorzunehmen und dabei den begrenzten Bildschirmplatz des Mobilgerätes bestmöglich ausnutzt. Hierbei muss die Struktur des Hostgerätes nicht zwangsweise imitiert werden. Sie sollte aber der Knotenansicht aus der Übersicht farblich gleichen, um den Benutzer nicht zu verwirren.

Wechsel zwischen beiden Ansichten Es muss ein fließender Wechsel zwischen beiden Ansichten möglich sein, mit der Vorgabe, dass die Übersicht der gerade bearbeiteten Stelle nicht verloren geht. Um dies zu erreichen, werden visuelle Merkmale benutzt. Kanten sollen in der Detailansicht an der gleichen Position wie in der Übersichtsansicht an den Knoten angeschlossen sein. Optional sollen die Kanten entsprechend ihrer Position eingefärbt werden, so dass ein Color-Matching erfolgen kann.

Einbezug des Knotenkontext Der Knotenkontext muss zur Anzeige definiert werden. Dieser umfasst im vorgestellten Konzept alle zum Knoten gehörigen Rezeptoren und Emitter, sowie an diese angeschlossene Kanten.

Verfolgung einer Kante Es muss eine einfache Aktion definiert werden, um dem Graphenfluss einer Kante folgen zu können. Diese soll möglichst an die allgemeinen Bedienmustern von Smartphones angepasst sein.

Darüber hinaus sind folgende Funktionen zur kollaborativen Arbeit denkbar:

Anzeige der Nutzerposition eines Mobilgeräts Die Hostapplikation könnte stets oder bei einem Zustandswechsel den Viewport des Smartphone-Nutzers anzeigen, damit dieser sich orientieren kann und seine Partner wissen, welche Stellen er gerade bearbeitet. Ein gerade in Bearbeitung befindlicher Knoten könnte optisch hervorgehoben werden.

Ping auf der Übersichtsansicht Ein Benutzer soll in der Lage sein, auf der Übersichtsansicht eine Art Ortsmarkierung zu setzen, welche dann auf dem Hostgerät an der entsprechenden Stelle sichtbar ist. Umgekehrt soll dies ebenso möglich sein, um dem Benutzer des Mobilgerätes zu zeigen, wohin er navigieren muss.

Sprung zu markierter Stelle Um die vorige Funktion zu erweitern, könnte man den Benutzer des Mobilgerätes durch eine Markierung direkt in die Detailansicht des Knotens wechseln lassen. Dies nimmt dem Benutzer jedoch einen Teil der Autonomie bei der Bedienung des Gerätes und wird somit vorerst nicht in Betracht gezogen.

Das umzusetzende Konzept verzichtet absichtlich auf die Implementierung einer totalen Zoom-Funktionalität, da diese nach den aufgestellten Hypothesen aus Kapitel 5.2 auf einem Mobilgerät zu einer umständlichen Navigation führt. Ein ständiges Zoomen und Verschieben der Ansicht soll vermieden werden.

5.1.1 Realisierung der Übersichtsansicht

Zur Realisierung der Übersichtsansicht wird die Struktur des gesamten Graphen auf dem Mobilgerät angezeigt. Knotendetails, wie Emitterwerte oder erweiterte Inhalte, werden in dieser Ansicht nicht oder nur teilweise dargestellt.

Die Struktur des Graphen wird hierbei an die des Hostprogrammes angepasst. Dies fördert das visuelle Verständnis des Graphen beim Wechsel zwischen Host- und Mobilgerät. Um dies zu ermöglichen, werden die Koordinaten der Knoten aus dem Graphmodell des Hosts übernommen. Dies sorgt für das richtige Verhältnis der Knoten zueinander. Eventuell müssen die Abstände der Knoten zueinander, aufgrund der kleineren Objektgröße, proportional skaliert werden.

Die Darstellung der Kanten erfolgt entsprechend der Filter/Flow-Metapher. Kanten werden hierbei blau eingefärbt, die Dicke der Kanten entspricht der verbleibenden Filtermenge nach jedem Filterknoten. Diese wird relativ zur zugehörigen Ausgangsmenge berechnet, sofern diese vorliegt. Die Dicke der Kanten wird anschließend noch passend zur Gesamtansicht skaliert. Existiert nur die Struktur und keine Ausgangsmengen, so werden alle Kanten gleich dick dargestellt.

Um dem Aspekt der Übersicht beim Wechsel zwischen den Ansichten Rechnung zu tragen, sollen sich Kanten entsprechend ihrer Reihenfolge von links nach rechts durch visuelle Merkmale unterscheiden. Somit kann eine Zuordnung an die Konnektoren des Knotens in der Detailansicht erfolgen. Die Kanten eines Knotens werden hierzu in verschiedenen Farben dargestellt. Die verwendeten Farben können somit pro Knoten wiederverwendet werden. Um die Fiter/Flow-Metapher zu erhalten, sollen verschiedene Blautöne verwendet werden.

Dem Benutzer wird zur Navigation ein kleines Rechteckfenster zur Verfügung gestellt, welches auf der Übersichtsansicht mittels Berührung an die gewünschten Position geschoben werden kann. Das Rechteckfenster folgt bei Berührung des Displays dem Finger des Benutzers und verbleibt an der Position, an dem die Berührung endet. Es wird der Knoten farbig markiert, welcher am nächsten am Mittelpunkt des Rechtecks liegt. Der Benutzer kann anschließend über einen Knopf zur Detailansicht wechseln, um den gewählten Ausschnitt zu betrachten. Hierbei wird der farbig markierte Knoten als Ausgangspunkt fixiert.

Der Benutzer muss zum Bearbeiten eines Knotens in die Detailansicht wechseln. Um zu einem anderen Punkt im Graphen zu navigieren, kann er entweder dem Fluss einer Kante folgen oder über einen Knopf zurück zur Übersicht springen.

Abbildung 5.1 zeigt den Entwurf der Übersichtsansicht.

5.1.2 Realisierung der Detailansicht

Die Detailansicht lehnt an den in Kapitel 4.1.2 beschriebenen Ansatz an. Ein ausgewählter Knoten soll untersucht und bearbeitet werden. Hierzu wird er in der Detailansicht

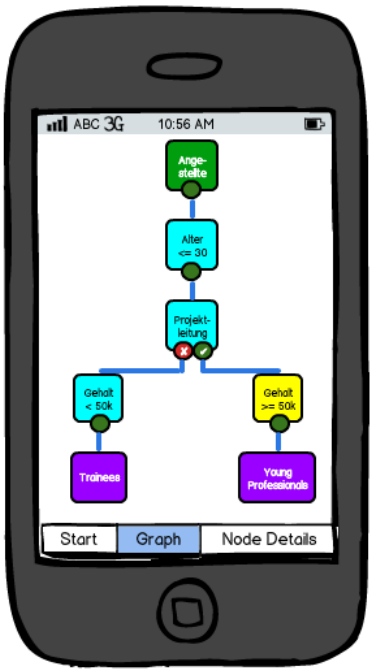


Abbildung 5.1: Mockup der Minimap

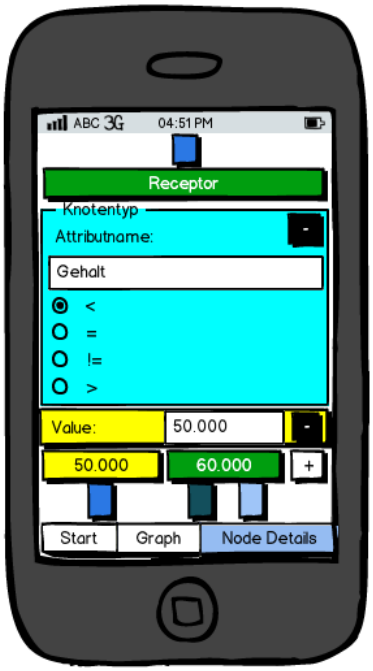


Abbildung 5.2: Mockup der Detailansicht

zentriert und inklusive seiner Attribute und Konnektoren dargestellt. Eingehende und ausgehende Konnektoren werden als eine Art Tab am oberen und unteren Rand des Knotens dargestellt.

Die Farbe des Knoteninhalts, sowie der Konnektoren, entspricht denen aus der Übersicht. Eingehende und ausgehende Kanten werden entsprechend der Filter/Flow-Metapher, blau eingefärbt und entsprechend ihrer Ergebnismengen dicker und dünner gezeichnet. Analog zur Übersichtsansicht in Kapitel 5.1.1 werden die Kanten entsprechend ihrer Reihenfolge in verschiedenen Farbtönen eingefärbt, um die Orientierung zu vereinfachen. Die Farbtöne müssen hierbei mit denen aus der Übersichtsansicht übereinstimmen, um eine Zuordnung zu ermöglichen.

Abbildung 5.2 zeigt den Entwurf dieser Detailansicht.

Knoteninhalt

Der Inhalt eines Knotens ist vom Knotentyp abhängig und wird somit entsprechend der Anforderungen implementiert. Jeder Detailknoten verfügt jedoch über einen Knotentyp, der links oben als Überschrift den Knoten identifiziert. Des Weiteren gibt es in der rechten oberen Ecke einen Knopf zum Löschen des Knotens aus dem Graphen. Die weitere Ansicht ist abhängig von den Eigenschaften des Knotentyps. Vergleichsknoten besitzen beispielsweise ein Attributfeld, welches über eine Texteingabebox repräsentiert wird, sowie einen Operator, der über eine Auswahl an Radio-Buttons gesetzt werden kann. Weitere Eigenschaften anderer Knotentypen werden mit einem passenden visuellen Bedienelement repräsentiert.

Existieren mehr Elemente, als auf den Bildschirm passen, so können die visuellen Elemente in einer vertikal scrollbaren Liste innerhalb des Knoteninhalt-Bereichs dargestellt werden.

Konnektoren

Die Konnektoren werden am oberen und unteren Ende des Knoteninhalts in einem horizontalen Band dargestellt. Jeder Konnektor ist groß genug, um Informationen über seinen Wert anzuzeigen und gleichzeitig gut über eine Berührung auswählbar zu sein. Er darf somit eine bestimmte Mindestgröße, auch ohne Wertanzeige, nicht unterschreiten. Über einen Button am rechten Rand können beliebig viele Konnektoren hinzugefügt werden, sofern der Knotentyp dies unterstützt. Wird eine bestimmte Anzahl an Konnektoren überschritten, kann das Konnektorband zur vollständigen Darstellung aller Konnektoren horizontal gescrollt werden.

Um einen Konnektor zu bearbeiten muss dieser mit dem Finger berührt werden. Direkt über dem Konnektorband erscheint nun der Konnektorinhalt des gewählten Knotens. Dieser aktualisiert sich, sobald ein anderer Knoten ausgewählt wird. Die Verwendung entspricht somit der von Tabs. Der Knoteninhalt kann wiederum ein Textfeld, eine

Checkbox oder komplexere Eingabefelder enthalten, um die Werte der Konnektoren ansprechend zu repräsentieren und vom Benutzer ändern lassen zu können. Am rechten Rand des Knoteninhalts befindet sich ebenfalls ein Button zum Löschen des Konnektors.

Am unteren Rand eines Konnektors befinden sich alle an ihn angeschlossenen Flows. Diese sollen mit einem kurzen vertikalen Hinweistext versehen werden, der Aufschluss über den Zielknoten gibt. Hier kann beispielsweise ein Akronym des Knotentyps verwendet werden. Werden an einen Konnektor mehr Kanten angeschlossen, als über die verfügbare Breite des Konnektor-Elements dargestellt werden kann, vergrößert sich dieses entsprechend. Beim Löschen werden automatisch alle an diesen Konnektor angeschlossenen Kanten mit gelöscht.

Bedienung

Attribute können nun über Berührung bearbeitet werden. Neue Attribute können über einen „+“-Button hinzugefügt werden. Die Bearbeitung von Kanten erfolgt analog. Um einen Konnektor mit Hilfe einer Kante an einen entfernten Knoten anzuschließen, wird in diesem Fall eine Liste der Knoten in der Umgebung angezeigt. Das Ziehen von Kanten über die gesamte Breite des Graphen kann nur über Listen realisiert werden. Als weitergehende Fragestellung kann untersucht werden, ob dieser Anwendungsfall überhaupt notwendig ist. Es ist auch denkbar, dass diese Funktion der Hostinstanz vorbehalten bleibt.

Alternativ soll die Möglichkeit geboten werden, sich Konnektoren in einer Liste zu merken, um anschließend Kanten am Zielpunkt erstellen zu können. Durch langes Berühren eines Konnektors soll dieser zu einer temporären Liste hinzugefügt werden. Der Nutzer kann auf diese Möglichkeit mehrere Konnektoren speichern. Nun kann der Nutzer zum Zielkonnektor wechseln, und die Konnektoren zum Verbinden aus der gespeicherten Liste wählen. Das Prinzip lehnt sich somit an das bekannte Copy & Paste aus den Dateimanagern gängiger Betriebssysteme an.

Folgen von Kanten durch Handgesten

Um einer Kante zu folgen kann der Benutzer auf die Darstellung der jeweiligen Kante klicken, um zum Zielkonnektor zu gelangen. Die Ansicht wechselt daraufhin in die Detailansicht des Zielknotens.

Da sich die Bedienung der schmalen Flow-Elemente voraussichtlich als schwierig erweist, wird eine weitere Möglichkeit zur Verschiebung des Viewports eingeführt. Es wird eine gestenbasierte Steuerung verwendet, die auf allen gängigen Smartphone-Betriebssystemen eingesetzt wird und mit einem Touchscreen leicht zu bedienen ist [2]. Der Benutzer berührt hierbei die angezeigte Arbeitsoberfläche und zieht diese in die gewünschte Richtung. Die Arbeitsoberfläche bewegt sich nun in die gewünschte Richtung. Dies simuliert das physikalische Verhalten der dargestellten Oberfläche. Der Viewport, beziehungsweise Bildschirmausschnitt, verschiebt sich also in entgegengesetzter Richtung. In einer weiteren

Variante schiebt der Benutzer den Finger nach Absetzen schnell in eine Richtung, um ein Anstoßen zu simulieren. Je nach Implementierung der Arbeitsoberfläche kommt diese daraufhin erst nach einiger Zeit zum Stehen.

In der beschriebenen Detailansicht wird dieses Anstoß-Konzept benutzt, um den aktuell angezeigten Knoten zu wechseln, also einer Kante in eine gewünschte Richtung zu folgen. Hierbei kann die Anstoß-Bewegung sowohl in vertikaler als auch horizontaler Richtung erfolgen. Es werden die folgenden Fälle unterschieden:

- Stößt der Benutzer die Arbeitsfläche nach oben an, so wird die erste Kante des ausgewählten Emitters verfolgt, sofern eine Kante vorhanden ist. Andernfalls soll eine Hinweismeldung ausgegeben werden. Ist nur ein Emitter vorhanden wird dieser automatisch als ausgewählt betrachtet. Dies entspricht einer Und-Verknüpfung beider Knoten.
- Stößt der Benutzer die Arbeitsfläche nach unten an, so wird die erste Kante des ersten Rezeptors verfolgt, sofern diese vorhanden ist, und der Vorgängerknoten keinen Parameter für den derzeitigen Knoten darstellt. Dies entspricht einer Und-Verknüpfung beider Knoten.
- Ein Anstoßen nach links zeigt einen direkten linken Nachbarn des derzeitigen Knotens an. Für beide Knoten gilt, dass sie den selben Vorgängerknoten besitzen müssen. Dies entspricht einer Oder-Verknüpfung beider Knoten.
- Ein Anstoßen nach rechts zeigt einen direkten rechten Nachbarn des derzeitigen Knotens an. Für beide Knoten gilt, dass sie den selben Vorgängerknoten besitzen müssen. Dies entspricht einer Oder-Verknüpfung beider Knoten.

Ist keine passende Kante vorhanden, der die Arbeitsfläche folgen kann, so wird dem Benutzer eine entsprechende Hinweismeldung angezeigt.

Durch die Gesten-Navigation kann der Benutzer schnell zwischen benachbarten Knoten wechseln und logische Abhängigkeiten betrachten. Er kann beispielsweise erkennen, dass zwei Bedingungen gleichzeitig zutreffen müssen, indem er nach oben oder unten wischt. Ebenfalls kann er überprüfen, ob andere Bedingungen erfüllt werden können, indem er die Arbeitsoberfläche horizontal bewegt.

Um dem Benutzer einen Indikator zu geben, ob es einen Nachbarknoten gibt, kann an den Seiten des derzeitigen Knotens eine Pfeilmarkierung eingeblendet werden, die angibt, ob eine Navigationsmöglichkeit besteht.

Abbildung 5.4 soll die Funktion der Gestensteuerung anhand der Minimap verdeutlichen. Bei dem vorliegenden Fall hat der Benutzer den gelben Knoten ausgewählt und befindet sich in dessen Detailansicht. Die Pfeile 1 bis 4 (im Uhrzeigersinn beginnend mit oben) zeigen auf die Knoten, die durch das Wischen in die jeweilige Richtung vom markierten Knoten aus erreicht werden können. Das Wischen nach Links, repräsentiert durch die 4, ist nicht möglich und würde eine Hinweismeldung werfen.

5.1.3 Realisierung des Anwendungseinstiegspunktes

Die in Kapitel 5.1.1 und Kapitel 5.1.2 beschriebenen Darstellungen sollen nun sinnvoll verbunden werden. Voraussetzung ist, dass der Benutzer zwischen beiden Ansichten wechseln kann, ohne dass die Übersicht über den gerade bearbeiteten Ausschnitt verloren geht. Der angezeigte Detailknoten muss somit immer dem ausgewählten Knoten aus der Strukturansicht des Graphen entsprechen. Umgekehrt muss der angezeigte Detailknoten in der Strukturansicht ausgewählt, beziehungsweise markiert werden.

Um den Ansicht-Wechsel zu ermöglichen, soll eine Tab-Leiste verwendet werden. Diese soll am unteren Bildschirmrand eingeblendet werden. Durch die Nutzung der Tabs ist es möglich, jederzeit schnell zwischen beiden Ansichten zu wechseln.

Zusätzlich zu den beiden Ansichten des Graphen wird eine Einstellungsansicht als Tab hinzugefügt. Hier kann der Nutzer einen Graphen laden und speichern, sowie die Arbeitsfläche leeren um einen neuen Graphen zu erstellen. In nachfolgenden Implementierungen soll der Benutzer hier die Verbindung mit einer Hostinstanz aufnehmen und trennen können. Die Tabs sollen in sinnvoller Reihenfolge angeordnet werden, so dass sie den Arbeitsfluss am besten widerspiegeln. Aus dem Arbeitsfluss ergibt sich folgende Reihenfolge, die ebenfalls eine Konkretisierung der Arbeitsschritte darstellt:

1. Einstellungsansicht
2. Strukturansicht des Graphen
3. Detailansicht des Graphen

Abbildung 5.3 zeigt den Entwurf der Einstellungsansicht.

5.2 Hypothesen zur Bedienbarkeit

Die folgenden Hypothesen beruhen auf den Problemstellungen aus Kapitel 2.2.3. Es sind Annahmen darüber, warum das entwickelte Konzept besser zur Darstellung von Graphen auf mobilen Endgeräten geeignet ist, als die direkte Nutzung eines bereits vorhandenen Konzeptes aus 4.1.

1. Durch die geteilte Darstellung von Graphenstruktur und -Details können beide Aspekte vollständig abgedeckt werden, ohne Einbußen in der Bedienung hinnehmen zu müssen. Konkret bedeutet dies, dass die Struktur des ganzen Graphen zur schnellen Navigation und Orientierung darstellbar ist und gleichzeitig über die Detailansicht alle Details eines Knotens einsehbar und veränderbar sind.
2. Die schnelle Navigation ist durch den Einsatz der Strukturansicht, sowie dem jederzeit möglichen Wechsel zwischen Struktur- und Detailansicht über die Tabs, gegeben. Der Nutzer kann jeden beliebigen Knoten rasch erreichen und bearbeiten (siehe Kapitel 5.1.3).

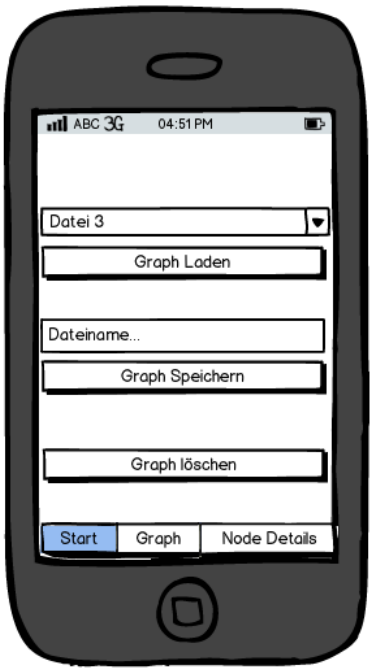


Abbildung 5.3: Mockup der Einstellungen: Ermöglicht Laden und Speichern des Graphen, sowie Leeren der Arbeitsoberfläche.

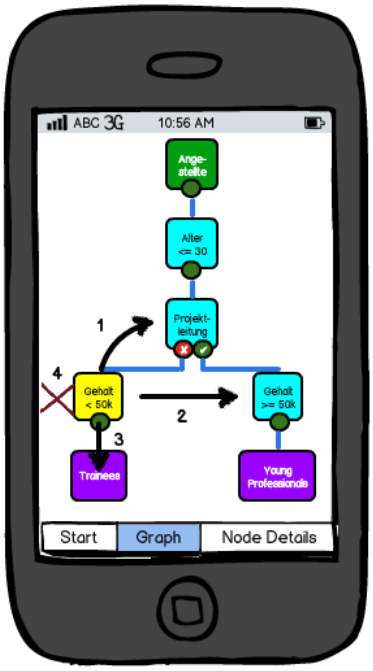


Abbildung 5.4: Mockup der Gestensteuerung: (1) Wischen nach oben, (2) Wischen nach rechts, (3) Wischen nach unten, (4) Wischen nach links.

3. Die Übersichtlichkeit geht beim Wechsel zwischen Hostapplikation und Clientapplikation nicht verloren. Die Strukturansicht spiegelt die Struktur des Graphens auf dem Host wider und ermöglicht so die kooperative Arbeit. Beim Wechsel zwischen Struktur- und Detailansicht helfen eingefärbte Kanten dem Nutzer bei der Orientierung.
4. Die Detailansicht dient der einfachen Bedienung über den berührungsempfindlichen Bildschirm. Bedienelemente sind genügend groß, damit sie vom Benutzer leicht betätigt werden können. Das gestenbasierte Bedienkonzept unterstützt die intuitive Bedienung und beschleunigt das Arbeiten mit dem Detailgraphen.
5. Die verwendeten Abläufe zur Durchführung einer Interaktion aus Kapitel 5.3 sind intuitiv und nachvollziehbar. Die Fehleranfälligkeit bei der Durchführung ist gering.

5.3 Use-Cases

Im Folgenden werden die im Prototyp möglichen Interaktionen als Use-Cases festgehalten und beschrieben. Ein Use-Case beschreibt Arbeitsschritte aus Sicht des Benutzers, die nötig sind, um eine Handlung durchzuführen und das beschriebene Ziel zu erreichen. Diese werden anhand des implementierten Konzeptes beschrieben. Als Akteure dienen der Benutzer und die zu entwickelnde Smartphone-Applikation.

5.3.1 Graph laden

Use-Case	***Graph laden***
Ziel	Der Benutzer möchte einen Graph zur Bearbeitung laden.
Vorbedingungen	• Es ist eine Repräsentation eines Graphen als Datei auf dem Smartphone vorhanden.
Regulärer Ablauf	1. Der Benutzer wählt den Reiter *Einstellungen* aus 2. Der Benutzer öffnet die Datei-Liste 3. Die Client-Applikation zeigt alle auf dem Gerät vorhandenen Dateien an 4. Der Benutzer wählt eine Datei aus der Liste aus 5. Der Benutzer klickt auf „Graph laden“ 6. Die Applikation lädt die Datei und zeigt den Graph an
Sonderfälle	4. Der Graph kann nicht geladen werden, da die Datei nicht den Vorgaben entspricht
Nachbedingung	Die ausgewählte Datei wurde geladen und der enthaltene Graph wird angezeigt.

5.3.2 Graph speichern

Use-Case	***Graph speichern***
Ziel	Der Benutzer möchte einen Graph speichern.
Vorbedingungen	• Es wird ein Graph angezeigt.
Regulärer Ablauf	1. Der Benutzer wählt den Reiter *Einstellungen* aus 2. Der Benutzer wählt einen Dateinamen über das Eingabefeld aus 3. Der Benutzer klickt auf „Graph speichern“ 4. Die Applikation speichert den Graphen auf der SD-Karte
Sonderfälle	4. Der Graph kann auf Grund fehlender Schreibrechte nicht gespeichert werden 4. Der Dateiname ist bereits vorhanden, die Client-Applikation fragt den Nutzer, ob er überschrieben werden soll
Nachbedingung	Die Datei wurde unter dem angegebenen Dateinamen gespeichert.

5.3.3 Knoten auswählen

Use-Case	***Knoten auswählen***
Ziel	Der Benutzer möchte einen Knoten zum Bearbeiten auswählen.
Vorbedingungen	• Es wird ein Graph angezeigt.
Regulärer Ablauf	1. Der Benutzer berührt den Bildschirm. Das Auswahlrechteck folgt nun der Berührung des Benutzers 2. Der Benutzer verschiebt das Auswahldreieck zum gewünschten Knoten. Der Knoten im Zentrum des Auswahlrechtecks wird farblich hervorgehoben 3. Der Benutzer hebt den Finger vom Bildschirm 4. Der ausgewählte Knoten bleibt markiert
Sonderfälle	
Nachbedingung	Der gewünschte Knoten im Mittelpunkt des Auswahlrechtecks wurde ausgewählt und wird hervorgehoben.

5.3.4 Knoten bearbeiten

Use-Case	***Knoten bearbeiten***
Ziel	Der Benutzer möchte einen Knoten bearbeiten.
Vorbedingungen	• Es wird ein Graph angezeigt. • Es wurde ein Knoten ausgewählt, siehe 5.3.3
Regulärer Ablauf	1. Der Benutzer klickt auf den Reiter *Detailansicht* 2. Der Benutzer wählt das Element aus, das er bearbeiten möchte 3. Der Benutzer ändert den Wert über die erscheinende kontextabhängige Eingabe
Sonderfälle	3. Das Element kann auf Grund fehlender Rechte nicht bearbeitet werden
Nachbedingung	Das gewünschte Knoten-Element wurde bearbeitet.

5.3.5 Knoten hinzufügen

Use-Case	***Knoten hinzufügen***
Ziel	Der Benutzer möchte einen Knoten zum Graphen hinzufügen.
Vorbedingungen	• Es wird ein Graph angezeigt. • Der Reiter *Strukturansicht* ist ausgewählt.
Regulärer Ablauf	1. Der Benutzer berührt den Bildschirm für mindestens 2 Sekunden 2. Es wird ein Menü eingeblendet zur Auswahl des gewünschten Knotentyps 3. Der Benutzer wählt den gewünschten Knotentyp aus der Liste 4. Der Benutzer klickt auf „Knoten einfügen“ 5. Es wird ein neuer Knoten vom gewählten Knotentyp an der Berührungsposition eingefügt
Sonderfälle	2. Der Graph kann auf Grund fehlender Rechte nicht verändert werden. Es erscheint eine Hinweismeldung
Nachbedingung	Der gewählte Knotentyp wurde an der Berührungsposition eingefügt. Die Applikation zeigt den Knoten zur Bearbeitung in der Detailansicht an.

5.3.6 Knoten löschen

Use-Case	***Knoten löschen***
Ziel	Der Benutzer möchte einen Knoten aus dem Graph löschen.
Vorbedingungen	• Es wird ein Graph angezeigt.
Regulärer Ablauf	1. Der Benutzer wählt den gewünschten Knoten aus, siehe 5.3.3 2. Der Benutzer klickt auf den Reiter „Detailansicht“ 3. Der Benutzer klickt auf den Knopf zum Löschen des Knotens 4. Die Applikation fragt den Benutzer, ob der Knoten wirklich gelöscht werden soll 5. Der Benutzer bestätigt den Löschvorgang 6. Der Knoten samt, seiner Attribute und Verknüpfungen, wird aus dem Graph gelöscht
Sonderfälle	3. Der Graph kann wegen fehlender Rechte nicht verändert werden. Es wird kein Symbol zum Löschen eingeblendet 5. Der Benutzer klickt auf „Abbrechen“. Der Vorgang wird abgebrochen
Nachbedingung	Der Graph wird in der Übersicht angezeigt. Der gewählte Knoten wurde aus dem Graph gelöscht.

5.3.7 Konnektor hinzufügen

Use-Case	***Konnektor hinzufügen***
Ziel	Der Benutzer möchte einen Konnektor zu einem Knoten hinzufügen.
Vorbedingungen	• Es wird ein Graph angezeigt. • Der Benutzer hat einen Knoten zur Bearbeitung ausgewählt, siehe 5.3.4 • Der Reiter *Detailansicht* ist ausgewählt
Regulärer Ablauf	1. Der Benutzer berührt den „+“-Knopf zum Hinzufügen eines Konnektors 2. Es wird ein neuer Konnektor angelegt
Sonderfälle	1. Der Knotentyp lässt das Hinzufügen von Emittern oder Rezeptoren nicht zu. In diesem Fall wird an der entsprechenden Stelle kein „+“-Knopf angezeigt
Nachbedingung	Es wurde ein neuer Konnektor zum Knoten hinzugefügt. Der Konnektor wurde, zur Bearbeitung ausgewählt.

5.3.8 Konnektor entfernen

Use-Case	***Konnektor entfernen***
Ziel	Der Benutzer möchte einen Konnektor von einem Knoten entfernen.
Vorbedingungen	• Es wird ein Graph angezeigt. • Der Benutzer hat einen Knoten zur Bearbeitung ausgewählt, siehe 5.3.4 • Der Reiter *Detailansicht* ist ausgewählt
Regulärer Ablauf	1. Der Benutzer wählt den gewünschten Konnektor aus 2. Der Benutzer klickt auf den „-“-Knopf innerhalb des Konnektorinhalts 3. Die Applikation fragt den Benutzer, ob er den Konnektor entfernen möchte 4. Der Benutzer bestätigt das Entfernen des Konnektors 5. Der gewählte Konnektor wird entfernt. Es werden zudem alle mit ihm verknüpften Kanten entfernt
Sonderfälle	2. Der Knotentyp lässt das Entfernen von Emittern oder Rezeptoren nicht zu. Es wird kein Knopf zum Entfernen angezeigt 4. Der Benutzer wählt „Abbrechen“. Der Vorgang wird abgebrochen
Nachbedingung	Der gewählte Konnektor wurde vom Knoten entfernt.

5.3.9 Kante in Detailansicht verfolgen

Use-Case	***Kante in Detailansicht verfolgen***
Ziel	Der Benutzer möchte in der Detailansicht eine Kante verfolgen.
Vorbedingungen	• Es wird ein Graph angezeigt. • Der Reiter *Detailansicht* ist ausgewählt • Der Benutzer hat einen Knoten zur Bearbeitung ausgewählt, siehe 5.3.4
Regulärer Ablauf	1. Der Benutzer berührt die gewünschte Kante kurz 2. Die Anzeige wechselt zur Detailansicht des Zielknotens der Kante
Sonderfälle	
Nachbedingung	Es wird die Detailansicht des Zielknotens angezeigt.

5.3.10 Kante hinzufügen

Use-Case	***Kante hinzufügen***
Ziel	Der Benutzer möchte eine Kante hinzufügen.
Vorbedingungen	• Es wird ein Graph angezeigt • Der Benutzer hat einen Knoten zur Bearbeitung ausgewählt, siehe 5.3.4 • Der Reiter *Detailansicht* ist ausgewählt
Regulärer Ablauf	1. Der Benutzer klickt auf den „Verbinden“-Knopf des gewünschten Start-Konnektors 2. Der Konnektor wird zum Verknüpfen gemerkt 3. Die Applikation wechselt in die Graph-Übersicht 4. Der Benutzer wählt den gewünschten Ziel-Knoten aus 5. Der Benutzer wechselt in die Detailansicht 6. Der Benutzer klickt auf den „Anschließen“-Knopf am gewünschten Konnektor 7. Es wird eine Kante zwischen Start- und Ziel-Konnektor erstellt
Sonderfälle	1. Es können keine Kanten im Graph erstellt werden. Es wird kein „Verbinden“-Knopf angezeigt. 6. Die Kante kann nicht am Ziel-Konnektor angeschlossen werden. Es wird kein Knopf zum Anschließen angezeigt.
Nachbedingung	Es wurde eine Kante zwischen den gewählten Konnektoren hinzugefügt.

5.3.11 Kante entfernen

Use-Case	***Kante entfernen***
Ziel	Der Benutzer möchte eine Kante entfernen.
Vorbedingungen	• Es wird ein Graph angezeigt • Der Benutzer hat einen Knoten zur Bearbeitung ausgewählt, siehe 5.3.4 • Der Reiter *Detailansicht* ist ausgewählt
Regulärer Ablauf	1. Der Benutzer berührt die gewünschte Kante für mindestens zwei Sekunden 2. Die Applikation fragt den Benutzer, ob er die Kante löschen möchte 3. Der Benutzer bestätigt das Entfernen der Kante 4. Die Kante wird aus dem Graphen entfernt
Sonderfälle	2. Der Graph kann wegen fehlender Rechte nicht verändert werden. Es wird eine Hinweismeldung angezeigt, dass die Kante nicht gelöscht werden kann 3. Der Benutzer klickt auf „Abbrechen“, der Vorgang wird abgebrochen
Nachbedingung	Die gewählte Kante wurde gelöscht. Die Applikation zeigt die Graph-Übersicht an.

5.3.12 Optionale Use-Cases

Die im Folgenden aufgeführten Use-Cases sind als optional zu betrachten und müssen nicht im Rahmen der Diplomarbeit umgesetzt werden. Sie beschreiben die zukünftige kollaborative Zusammenarbeit einer Client-Applikation mit der Host-Applikation. Als Akteure dienen somit der Benutzer selbst, die zu entwickelnde Smartphone-Anwendung als Client-Applikation, sowie die vorhandene Desktop-Anwendung als Host-Applikation.

5.3.13 Mit Host verbinden

Use-Case	***Mit Host verbinden***
Ziel	Der Benutzer möchte den Graphen von der Host-Applikation auf der Client-Applikation anzeigen.
Vorbedingungen	• Die Host-Applikation ist gestartet.
Regulärer Ablauf	1. Der Benutzer startet die Client-Applikation 2. Der Benutzer wechselt in die Start-Ansicht mit den Einstellungen 3. Der Benutzer gibt die IP-Adresse des Hosts in das vorgesehene Eingabefeld ein 4. Der Benutzer klickt auf „Verbinden“ 5. Der Graph der Host-Applikation wird auf dem Client angezeigt
Sonderfälle	4. a) Die Verbindung zur Host-Applikation kann nicht aufgebaut werden oder bricht unerwartet ab. Der Benutzer erhält eine Hinweismeldung und muss sich erneut mit dem Host verbinden. b) Es kommt zu Übertragungsfehlern beim Empfangen des Graphen. Die Client-Applikation fordert den übertragenen Graphen erneut an.
Nachbedingung	Solange die Verbindung besteht, zeigt der Client stets den aktuellen Graphen der Host-Applikation an. Der Knopf zum Verbinden wurde durch einen Knopf zum Trennen der Verbindung ersetzt.

5.3.14 Verbindung mit Host trennen

Use-Case	***Verbindung mit Host trennen***
Ziel	Der Benutzer möchte die Verbindung zur Host-Applikation trennen.
Vorbedingungen	• Die Host-Applikation ist gestartet. • Die Client-Applkation ist gestartet. • Es wurde eine Verbindung zu einer Host-Applikation aufgebaut, siehe 5.3.13
Regulärer Ablauf	1. Der Benutzer wechselt in die Start-Ansicht mit den Einstellungen 2. Der Benutzer klickt auf „Verbindung trennen“ 3. Die Verbindung zum Host wird getrennt. Es wird kein Graph mehr angezeigt.
Sonderfälle	
Nachbedingung	Die Verbindung zum Host wurde getrennt, es wird kein Graph mehr angezeigt. Der Knopf zum Trennen der Verbindung wurde durch einen Knopf zum Verbinden mit einem Host ersetzt.

6 Implementierung

Im Folgenden wird die technische Realisierung des Konzeptes dargestellt. Um den Anteil an Fremdsoftware gering zu halten, wird versucht, sich bei der Umsetzung auf die von Google bereitgestellte API zu beschränken.

6.1 Plattform

Die Client-Anwendung soll auf einem Mobilgerät implementiert werden. Es eignen sich somit die mobilen Betriebssysteme Android, iOS sowie Windows Phone 7.

iOS scheidet als Kandidat für die Entwicklung aus, da zum Test der Implementierung auf dem Endgerät eine iOS-Lizenz notwendig ist, welche jährlich verlängert werden muss, um die entwickelte App nutzen zu können. Dies liegt an der von Apple vorgeschriebenen Signierung von Apps. Zudem sind iPhones, welche auf iOS basieren, teurer als gleichwertige Smartphones basierend auf den Konkurrenzsystemen.

Windows Phone 7 käme wegen der Nutzung von C# als Programmiersprache als Entwicklungsgerät in Frage. Jedoch ist die Plattform noch recht neu und wenig verbreitet. Aufgrund dessen und der guten, frei zugänglichen Dokumentation von Google, wird Android als Betriebssystem ausgewählt.

Um eine gemeinsame Code-Basis für Desktop und Client-Anwendung zu schaffen, wurde im Vorfeld die Bibliothek „Mono for Android“ untersucht. Diese bietet die Möglichkeit, Android-Applikationen in Visual Studio 2010 und MonoDevelop mit der Programmiersprache C# zu schreiben, siehe Kapitel 6.2. Somit wurde es möglich, das Datenmodell und Bestandteile des Web Service wiederzuverwenden. Lediglich die Visualisierung, die im Fokus der Diplomarbeit steht, muss speziell für das Betriebssystem Android entworfen werden.

Die Verwendung von „Mono for Android“ macht es zudem möglich, zukünftige Plattformen leichter zu unterstützen. Windows Phone 7 verwendet ebenfalls C# als Programmiersprache und kann somit das Datenmodell, ohne Software von Drittanbietern, einbinden. Hierbei ist jedoch zu beachten, dass für Windows Phone 7 nicht der volle Funktionsumfang der .NET-Plattform zur Verfügung steht und gegebenenfalls Funktionen umgeschrieben werden müssen. Für iOS existiert, analog zu „Mono for Android“, „Mono Touch“, welches Apples Cocoa Touch API unter C# verfügbar macht.

6.2 Mono for Android

„Mono“ ist eine freie, quelloffene Implementierung von Microsofts .NET-Framework für die Systemumgebung Linux. Sie umfasst unter anderem einen Compiler für die Programmiersprache C#, sowie eine Implementierung der Common Language Runtime, welche zur Ausführung von .NET-Anwendungen benötigt wird. Mittlerweile existieren Versionen von Mono für viele weitere Plattformen, darunter Unix, OS X, Solaris, sowie seit einiger Zeit mobile Plattformen. Mono wurde ehemals von den Firmen Ximian und Novell im Jahre 2004 entwickelt. Durch die Übernahme von Novell durch Attachmate im April 2011, wurde im Mai 2011 die Firma *Xamarin* gegründet, die aus den ehemaligen Mono-Entwicklern besteht und somit die Projekte weiterführt.

„Mono for Android“, ehemals „Monodroid“, stellt eine Implementierung von Mono für Android dar. Sie wird proprietär von der Firma *Xamarin*, parallel zu „Mono Touch“, entwickelt und vertrieben. Somit ist eine einmalige Lizenz für die Nutzung des Übersetzers notwendig; es existieren jedoch Lizenzmodelle für Universitäten und Studenten. Das VIS-Institut stellt eine Lizenz bereit, die für die Entwicklung des Prototyps eingesetzt werden konnte.

Neue Versionen der nativen APIs werden schnellstmöglich umgesetzt. Somit unterstützt „Mono for Android“ die aktuelle Android-Version 4, Ice Cream Sandwich.

„Mono for Android“ besteht aus den folgenden Komponenten:

- Der Mono-Laufzeitumgebung
- Den Kern-.NET-Klassenbibliotheken, auch BCL (Base Class Library) genannt
- Den Bibliotheken, die die nativen Android- und Java-APIs binden
- SDK-Tools zum Packen, Bereitstellen und Debuggen von Anwendungen
- Eine Integration in Visual Studio 2010, für ferngesteuertes Bereitstellen und Debuggen

6.2.1 Vorteile von Mono

Die Verwendung von Mono for Android und dem angebotenen Visual Studio Plugin bietet die folgenden Vorteile gegenüber der nativen Programmierung in Java:

- Cross-Plattform-Entwicklung für alle aktuellen, mobilen Plattformen (Android, iOS, Windows Phone 7)
- Einbindung bestehender .NET-Bibliotheken in Android-Projekten
- Nutzbarkeit der aktuellen Android-APIs, da Xamarin Mono fortlaufend weiterentwickelt
- Unterstützung der x86-Emulator-Images, zur schnelleren Ausführung des Emulators auf x86-Prozessoren

Für erfahrene .NET-Entwickler bietet *Mono for Android* die folgenden Vorteile:

- Vollständige Unterstützung von LINQ, womit Arrays einfach abgefragt, sortiert und gefiltert werden können
- Unterstützung von Parallel LINQ, zum Schreiben und Ausführen von Parallelem Code auf mehreren Prozessorkernen

6.2.2 Lizenzmodell von Mono for Android

Mono for Android kann zur Implementierung und Test innerhalb des Emulators frei verwendet werden. Zum Test auf einem Endgerät und der kommerziellen Nutzung stehen laut [34] und [33] Lizenzen für Studenten, Akademiker, Selbstständige und Unternehmen zur Verfügung. Jede Lizenz umfasst hierbei die Veröffentlichung in App Stores, wie beispielsweise dem Play Store von Google, sowie ein Jahr Support in Form von Compiler-Updates.

6.3 Technische Einschränkungen

Durch die Verwendung von Android mit den aufgeführten Darstellungskonzepten aus den Kapiteln 2.4.4 und 2.4.3 ergeben sich folgende Einschränkungen:

- Die gleichzeitige Verwendung eines Canvas, zum freien Zeichnen, sowie eines Sichtenkonzepts, für interaktive Bedienelemente, wird für die Implementierung des Prototyps ausgeschlossen. Es ist zwar möglich, die `onDraw()`-Methode einer Sicht zu überschreiben, um eigene Funktionen auszuführen, widerspricht aber einer sauberen Sichtenarchitektur. Zudem wäre der Realisierungsaufwand gemessen an der prototypischen Implementierung unverhältnismäßig hoch.
- Das Zeichnen auf dem Canvas erfolgt immer vollständig. Dies gilt auch für Updates einzelner Teilbereiche. Eine bereichsensible Aktualisierung wird wegen der prototypischen Implementierung für zukünftige Verbesserungen vorbehalten.

Des Weiteren entstehen durch die Verwendung von Mono for Android die folgenden Einschränkungen hinsichtlich der Architektur des Prototyps:

- Ressourcen können zum Stand der Arbeit lediglich im Hauptprojekt integriert und referenziert werden. Somit müssen sich alle Klassen, die für die Darstellung auf XML-Dateien zurückgreifen, im Hauptprojekt befinden und können nicht in Bibliotheken ausgelagert werden. Ein Workaround ist möglich, würde aber zyklische Abhängigkeiten schaffen, die in jeder denkbaren Architektur unerwünscht sind.

6.4 Implementierung der Strukturansicht

Um die Strukturansicht aus Kapitel 5.1.1 umzusetzen, wird das Canvas-Konzept von Android verwendet. Die dazugehörige Klasse, die den Canvas bereitstellt, wird von der View-Klasse abgeleitet. Anschließend wird die `onDraw()`-Methode der Klasse überschrieben, um die eigene Darstellung des Graphen zu implementieren.

Die Klasse wird entsprechend Ihrer Funktion als „FlowDisplay“ benannt. Sie enthält den aktuellen Graphen als Datenmodell und kann jederzeit einen beliebigen anderen Graphen laden und darstellen. Die `onDraw()`-Methode iteriert über alle Knoten und Kanten im gegebenen Graphen und weist jedem Datenknoten den passenden virtuellen Knoten zu. Anschließend weist sie diese an, sich selbst mittels einer `draw()`-Methode darzustellen. Hierzu übergibt sie den Objekten alle Informationen, die zum Zeichnen notwendig sind, unter anderem den Canvas. Das Zeichnen der Knoten erfolgt über so genannte Drawables. Dies sind Zeichenobjekte, welche sowohl programmatisch, als auch mittels XML-Definition erstellt werden können. Der vorliegende Prototyp soll letztere Funktion verwenden, um die Darstellung der Elemente besser vom Code trennen zu können. Um Text innerhalb eines Zeichenobjektes darzustellen, wird die Android-Klasse `StaticLayout` verwendet. Diese gibt einen Container zur Formatierung und zum Umbruch von Textinhalten vor, welcher über den darzustellenden Knoten gelegt werden kann. Die Klasse `FlowDisplay` dient zusätzlich zum Abfangen und Verarbeiten der Berührungsinteraktionen des Benutzers.

6.4.1 Implementierung des Kanten-Algorithmus

Zum Zeichnen der Kanten beziehungsweise Flows in der Strukturansicht wird der Algorithmus von der Referenzimplementierung adaptiert und an die Zeichenumgebung von Android angepasst. Der Algorithmus dient zum Verbinden zweier Konnektoren durch einen gegebenen Flow. Ein Flow besteht hierbei aus drei Teilen – je zwei senkrechten Segmenten, die orthogonal an die Konnektoren angeschlossen sind, sowie einem Verbindungssegment, das die beiden Anschlusssegmente miteinander verbindet. Durch diese ist jederzeit sichtbar, an welchem Konnektor die jeweilige Kante angeschlossen ist. Beim Zeichnen des Verbindungssegments werden die folgenden zwei Fälle unterschieden:

Zielkonnektor unterhalb des Startkonnektors Dies ist der Standardfall einer Verbindung. Das Verbindungssegment besteht aus einer durchgängigen Kante, die die Anschlusssegmente auf direktem Weg miteinander verbindet, siehe Abbildung 6.1.

Zielkonnektor überhalb des Startkonnektors Die beiden Anschlusssegmente werden über eine orthogonale Verbindung miteinander verbunden. Hierzu wird das Verbindungssegment in drei Untersegmente aufgeteilt, die jeweils orthogonal zueinander stehen, siehe Abbildung 6.2.

Abbildungen 6.1 und 6.2 zeigen die beiden Darstellungsarten. Die Unterscheidung findet statt, damit direkte Kantenverbindungen beim Zeichnen nicht mit den Knoten die sie verbinden, kollidieren. Um die Kanten dennoch hinter den Knotenobjekten zu

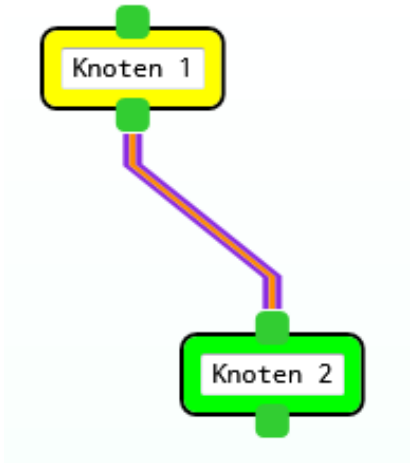


Abbildung 6.1: Flow-Zeichnen Fall 1: Direkte Verbindung

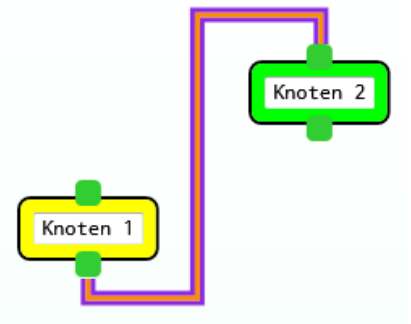


Abbildung 6.2: Flow-Zeichnen Fall 2: Orthogonale Verbindung

platzieren, damit diese die Darstellung anderer Knoten nicht behindern, wird der Kanten-Algorithmus stets vor dem Aufruf der `Draw()-Algorithmen` einzelner Knoten aufgerufen. Durch eine leichte Transparenz der Knotenhintergründe scheinen diese leicht durch den Knoten hindurch.

Der Algorithmus ermöglicht es weiterhin, Flows in eine beliebige Anzahl paralleler Segmente zu unterteilen. In den genannten Abbildungen sind die Flows in drei parallele Segmente unterteilt und durch die Farben Magenta und Orange visuell voneinander abgegrenzt.

6.5 Implementierung der Detailansicht

Um die Detailansicht aus Kapitel 5.1.2 umzusetzen, wird ein Android-Layout zur Anordnung aller enthaltenen Elemente verwendet. Dieses unterteilt den anzuzeigenden Knoten in drei Teilbereiche: Dem Header, zur Darstellung der Rezeptoren und eingehender Kanten, dem Content, zur Darstellung des Knoteninhalts und dem Footer, zur Darstellung der Emitter und ausgehenden Kanten. Alle Teilbereiche bestehen wiederum aus geschachtelten Layout-Klassen, um die enthaltenen Elemente passend anzuordnen.

Die Hintergründe der Layout- und View-Elemente werden als Drawables in XML realisiert und im Quellcode dem Background-Attribut zugewiesen. Darzustellende Knoteneigenschaften werden als interaktive Elemente, wie beispielsweise Listen und Buttons, innerhalb der Layouts angeordnet. Verfügbare Layouts finden sich in Kapitel 2.4.3. Für die Darstellung der Konnektoren wird beispielsweise eine *HorizontalScrollView*-Klasse verwendet, die es ermöglicht, beliebig viele Elemente in einer scrollbaren Liste darzustellen.

6.6 Implementierte Knotentypen

Der verwendete Prototyp orientiert sich an der vorhandenen Desktop-Implementierung für Filter/Flow-Graphen nach dem Konzept aus [14]. Hierbei werden teilweise Ausdrücke der Abfrage-Sprache *SPARQL* als Knoten abgebildet. Die folgenden Knotentypen sind im Prototyp umgesetzt:

Vergleichsoperatoren Vergleichen Werte vom Typ String, Integer, Date oder Boolean. Der Vergleichsoperator kann pro Knoten ausgewählt werden.

Between-Operator Prüft, ob sich ein Wert im vorgegeben Bereich befindet, der durch eine obere und eine untere Schranke definiert wird.

Predicate-Exists-Operator Prüft, ob das gegebene Prädikat existiert oder nicht.

Kommentar-Knoten Ein Knoten mit einem Textfeld, für Erläuterungen in einem Graphen. Kann auch als Platzhalter in Entwürfen verwendet werden.

Identity-Knoten Leitet die Identität der Eingangsmengen weiter und dient somit zur optischen Strukturierung eines Graphen.

6.7 Architektur des Prototyps

Die Architektur des Prototyp baut auf der Architektur der Referenzimplementierung für Desktop-Computer auf, siehe Unterabschnitt 4.2.2. Durch Kapselung von Komponenten in eigene Projekte wird ein hoher Wiederverwendungswert erzielt. Viele der genutzten Kernkomponenten können plattformübergreifend genutzt werden. So integriert der Prototyp das Datenmodell und die Datenstruktur der Referenzimplementierung unter Beibehaltung vorhandener Schnittstellen. Das visuelle Frontend muss jedoch für den Prototyp neu implementiert werden, sowie die Client-Seite der Web-Service-Kommunikation.

Um die Verwendung der Komponenten eindeutig zu kennzeichnen, werden die nachfolgenden Überschriften mit einem Label versehen. `[Einbindung]` sieht die Verwendung des unveränderten Quellcodes als Android-Projekt vor, weshalb Projekte dieser Art in der Entwicklungsumgebung mit `[And]` gekennzeichnet werden, um sie von den ursprünglichen Projekten zu unterscheiden. `[Referenzimplementierung]` gibt an, dass die Struktur des Projektes als Referenz für die analog zu entwickelnden Komponenten dient. `[Neuentwicklung]` kennzeichnet letztendlich alle Eigenentwicklungen des Diplomanden.

6.7.1 Architektur der Referenzimplementierung

Abbildung 9.1 im Anhang zeigt die Architekturübersicht der Referenzimplementierung. Sie enthält die im Folgenden beschriebenen Komponenten als eigenständige Projekte. Weiterhin ist ein Backend zur Ansteuerung von *SPARQL*-Datenbanken enthalten, das jedoch im Rahmen dieser Arbeit nicht besprochen wird.

FilterSolutions `[Bibliothek, Einbindung]`

`FilterSolutions` stellt das Datenmodell der Desktop, sowie der Mobilanwendung dar. Dieses definiert alle notwendigen Datentypen eines Graphen. Konkrete Anwendungsklassen erben von diesen abstrakten Klassen, beispielsweise die Klassen aus `DBFilter`, siehe Kapitel 6.7.1. Die folgenden Datentypen sind definiert:

`Graph` Das Datenmodell eines Graphen. Beherbergt Nodes und Flows.

`Node` Die Basisklasse jedes Knotens im Graphen

`Connector` Ein Anschluss an einen Knoten

`Emitter` Ein Knotenausgang, erbt von Connector

`Receptor` Ein Knoteneingang, erbt von Connector

`Flow` Eine Kante, die einen Emitter mit einem Receptor verbindet

`NodeCatalogue` Gibt Auskunft über die Knotentypen, die im gegebenen Anwendungsfall zur Instanziierung vorhanden sind. Wird beispielsweise in `DBFilter` abgeleitet und genutzt.

Des Weiteren sind Klassen zur Serialisierung dieser Basisklassen definiert:

`GraphSettings` Gerüst zum Speichern eines Graphen. Besteht aus mehreren `NodeSettings` und `FlowSettings`

`NodeSettings` Basisklasse für Knoteninformationen. Wird pro Knotentyp abgeleitet, beispielsweise in `DBFilter`

`FlowSettings` Speichert Informationen einer Kante

`NodeSettingsRegistry` Weist jedem Datenknoten den passenden Informationsknoten zu

`JSONHelper` Dient der Serialisierung von Informationsklassen in JSON, zur Speicherung und Übertragung von Daten

`FilterSolutions` ist unabhängig von der graphischen Darstellung des Graphen. Die Quelldateien werden im Prototyp unverändert als Android-Projekt eingebunden und als `FilterSolutions[And]` zur Verfügung gestellt.

FilterSolutions.Presentation [Bibliothek, Einbindung]

`FilterSolutions.Presentation` stellt eine Abstraktionsebene für die graphische Darstellung dar. Sie beinhaltet die folgenden Klassen, für die Darstellung von Graphelementen:

`VisualGraph` Eine visuelle Repräsentation eines Graphen

`VisualNode` Eine visuelle Repräsentation eines Knotens

`VisualNodeFactory` Dient der Zuordnung von visuellen Knoten auf vorhandene Knotentypen

`FilterSolutions.Presentation` ist ebenfalls unabhängig von der graphischen Darstellung auf der Zielplattform. Sie gibt vor, welche grundsätzlichen Methoden in den abgeleiteten visuellen Klassen implementiert werden müssen, um diese darzustellen. Die Quelldateien werden im Prototyp unverändert als Android-Projekt eingebunden und als `FilterSolutions.Presentation[And]` zur Verfügung gestellt.

DBFilter [Bibliothek, Einbindung]

`DBFilter` stellt einen Anwendungsfall von `FilterSolutions` dar, zur Visualisierung von Datenbankabfragen. Die Bibliothek beschreibt Knotenklassen basierend auf booleschen Ausdrücken, siehe Kapitel 2.1.2.

`DBFilterTerm` Ein datenbankspezifischer Filterknoten. Erbt von `Node` aus `FilterSolutons.Abstract`

`FilterCatalogue` Eine Spezialisierung von `FilterCatalogue`, zum Registrieren und Bereitstellen der implementieren Knotentypen

Basierend auf diesen allgemeinen Datentypen existieren Implementierungen aller benötigter Knotentypen aus Kapitel 2.1.5. Diese leiten sich von `DBFilterTerm` ab und implementieren die für den jeweiligen Knotentyp benötigten Attribute.

Die Bibliothek `DBFilter` ist unabhängig von der graphischen Darstellung und wird im Prototyp als `DBFilter[And]` unter Android eingebunden und bereit gestellt.

FilterSolutions.Presentation.WPF [Bibliothek, Referenzimplementierung]

Die *WPF*-Version der Präsentations-Bibliothek stellt in der Hierarchie die erste von der Oberfläche abhängige Implementierung dar. Sie dient zur Darstellung von Graphen mit Hilfe von *WPF*. Die enthaltenen Klassen spezialisieren die allgemeinen Versionen aus `FilterSolutions.Presentation` für die *WPF*-Umgebung. Es werden jedoch keine anwendungsspezifischen Merkmale für Knoten und Kanten festgelegt.

`WPFGraph` Ein von `VisualGraph` abgeleiteter Graph, der unter *WPF* dargestellt wird

`WPFNode` Ein von `VisualNode` abgeleiteter Knoten, der unter *WPF* dargestellt wird

`WPFConnectorElement` Stellt einen Konnektor unter *WPF* dar

`FlowElememt` Ein graphisches Element zur Darstellung eines Flows unter *WPF*

`DrawingContext` Ein an *WPF* angepasster Zeichenkontext, der unter Anderem den Canvas beherbergt

`FlowDisplay` Die eigentliche Graphen-Darstellungsklasse. Sie verfügt über Methoden zum Laden und Anzeigen eines Graphen. Sie bildet einen Graphen auf einen visuellen Graphen ab und bietet eine Factory zum Erstellen der visuellen Knoten an.

Der Aufbau dieses Projekts stellt die Grundlage für `FilterSolutions.Presentation.Android` dar, siehe Kapitel 6.7.2.

DBFilter.WPF `[Bibliothek, Referenzimplementierung]`

Das Projekt `DBFilter.WPF` stellt die visuelle Schnittstelle zu `DBFilter` unter *WPF* dar. Sie erbt einerseits von `DBFilter`, als auch von `FilterSolutions.Presentation.WPF`, deren Klassen sie weiter spezialisiert. Sie enthält visuelle Implementierungen jedes Knotentyps aus `DBFilter`, sowie folgende Basisklassen:

`VisualDBFilterTermNode` Ein Knoten zur Darstellung eines beliebigen `DBFilterTerm` Knotens

`DBFilterTermFactory` Eine Factory-Klasse zum Erstellen eines passenden visuellen Knotens zu einem gegebenen Datenknoten (`DBFilterTerm`)

Die implementierten Knotentypen aus Kapitel 6.6 spezialisieren `VisualDBFilterTermNode`. Um die visuellen Knoten den Datenknoten aus `DBFilter` zuzuordnen, existiert eine `DBFilterTermFactory`. In ihr werden alle verfügbaren Datenknotenklassen den visuellen Knotenklassen dieses Projekts zugeordnet.

Die Projektstruktur von `DBFilter.WPF` dient als Grundlage für `DBFilter.Android`, siehe Kapitel 6.7.2.

WPFFilterEditor `[Anwendung, Referenzimplementierung]`

Das Projekt `WPFFilterEditor` stellt das ausführbare Programm dar, um Graphen vom Typ `DBFilter` mittels *WPF* darzustellen. Es basiert auf den Bibliotheken `FilterSolutions`, sowie `DBFilter` und bindet `DBFilter.WPF`, sowie `FilterSolutions.Presentation.WPF` zur Visualisierung ein.

Das Programm bietet eine Oberfläche zum Anzeigen und Erstellen eines Graphen. Darüber hinaus existiert eine Menüleiste, mit deren Hilfe Graphen geladen und gespeichert

werden können. Zwischen Menüleiste und Zeichenoberfläche existiert noch eine Toolbar. Diese beherbergt alle Knotentypen zum Erstellen auf der Zeichenoberfläche.

`WPFFilterEditor` stellt somit ein Gerüst dar, das die `FlowDisplay`-Klasse der verwendeten Präsentations-Bilbiothek `FilterSolutions.Presentation.WPF` einbindet.

6.7.2 Architektur unter Android

In Abbildung 9.2 ist die Architektur der Android-Projekte des Prototyps dargestellt. Aufgrund der technischen Einschränkungen aus Kapitel 6.3, müssen sich alle visuellen Klassen, die XML-Daten zur Darstellung verwenden, im Hauptprojekt befinden. Dies führt dazu, dass die Bibliothek, die die Implementierung der visuellen `DBFilter`-Knoten enthält, mit dem Hauptprojekt `AndroidFilterEditor` verschmolzen wird.

FilterSolutions.Presentation.Android `[Bibliothek, Neuentwicklung]`

Die Android-Variante der Präsentations-Bibliothek spezialisiert die abstrakten Präsentationsklassen aus `FilterSolutions.Presentation` für die Zeichenumgebung von Android. Sie bindet die Bibliothek `Android.Graphics` ein und implementiert beispielsweise den Android-Canvas. Die folgenden Klassen werden im Prototyp entwickelt:

`AndroidGraph` Darstellung eines Graphen unter Android. Spezialisiert `VisualGraph`.

`AndroidNode` Ein Knoten, der unter Android dargestellt werden soll. Spezialisiert `VisualNode`.

`FlowElement` Darstellung einer Kante unter Android.

`DrawingContext` Ein an `Android.Graphics` angepasster Zeichenkontext. Enthält den Canvas und Zeicheninformationen, wie XML-Ressourcen.

DBFilter.Android `[Integrierte Bibliothek, Neuentwicklung]`

Die Android-Erweiterung von `DBFilter`, die aus technischen Gründen als Quelltext in `AndroidFilterEditor` integriert ist. Sie stellt visuelle Knoten aller Knotenklassen aus `DBFilter` dar. Die Projektstruktur lehnt sich an `DBFilter.WPF` an, verwendet jedoch die `Android.Graphics`-Bibliothek statt der *WPF*-Bibliothek.

`DBFilter.Android` unterscheidet bei der Darstellung der Knoten und Kanten jedoch zwischen der Graph-Übersicht und der Detailansicht. Es existieren somit für jeden Knotentyp zwei visuelle Ausprägungen – eine Repräsentation, die sich mittels einer `Draw()`-Methode auf einem Canvas zeichnet, sowie eine Android-View, die aus interaktiven Elementen aufgebaut ist. Erstere sind hierbei durch das Präfix `Visual` gekennzeichnet, letztere durch `Detail`. Um die Knotentypen ihren entsprechenden Repräsentationen zuzuordnen existiert für jede Darstellungsart eine eigene Factory-Klasse.

Unter Anderem sind die folgenden Klassen definiert:

`VisualDBFilterTermNode` Ein Knoten zur strukturellen Darstellung eines beliebigen `DBFilterTerm` Knotens. Spezialisiert `AndroidNode`

`DetailDBFilterTermNode` Ein Knoten zur detaillierten Darstellung eines beliebigen `DBFilterTerm` Knotens. Spezialisiert `AndroidNode` und gibt eine View-Hierarchie des Detailknotens zurück

`VisualNodeType` Eine Klasse zur detaillierten Darstellung eines bestimmten Knotentyps mittels einer View-Hierarchie. Spezialisiert `VisualDBFilterTermNode`

`DetailNodeType` Eine Klasse zur strukturellen Darstellung eines bestimmten Knotentyps auf einem Canvas. Spezialisiert `Android.View`

`DBFilterTermDetailFactory` Eine Factory-Klasse zum Erstellen eines passenden visuellen Detailknotens zu einem gegebenen Datenknoten (`DBFilterTerm`)

`DBFilterTermStructureFactory` Eine Factory-Klasse zum Erstellen eines passenden visuellen Strukturknotens zu einem gegebenen Datenknoten (`DBFilterTerm`)

Die Darstellung von Kanten, Emittern und Rezeptoren erfolgt analog – es existieren sowohl `Visual`-Klassen zur Darstellung in der Übersicht, als auch `Detail`-Klassen zur Darstellung in einem Detailknoten, die auf interaktiven View-Elementen aufbauen.

AndroidFilterEditor `[Anwendung, Neuentwicklung]`

`AndroidFilterEditor` ist der Einstiegspunkt des Prototyps zur Darstellung und Änderung von Filter/Flow-Graphen auf Android. Das Projekt enthält die Klassen aus `DBFilter.Android` und bindet die Bibliotheken `FilterSolutions` und `DBFilter` als Datenmodell, sowie `FilterSolutions.Presentation` und `FilterSolutions.Presentation.Android` zur Darstellung ein. Zusätzlich wird die externe Bibliothek `SignalR` zur späteren Kommunikation mit der Hostanwendung eingebunden.

`AndroidFilterEditor` besteht aus vier Hauptbestandteilen beziehungsweise Activities. Die `MainActivity` stellt den Einstiegspunkt in die Anwendung dar und beherbergt die Tab-Leiste zur Steuerung der derzeitigen Ansicht. Die `SettingsActivity` dient zum Laden und Speichern, die `MinimapActivity` zeigt die Übersicht des derzeit geladenen Graphen und die `DetailNodeActivity` den derzeit ausgewählten Knoten der Übersicht im Detail. Alle vier Komponenten haben Zugriff auf eine zentrale `Settings`-Klasse, die den aktuellen Datengraphen, sowie den entsprechenden visuellen Graphen, bereit hält. Änderungen werden somit über die `Settings`-Klasse eingebracht, woraufhin diese alle lauschenden Komponenten über die gemachten Änderungen mit Hilfe von Events informiert.

`MainActivity` Der Einstiegspunkt in die Anwendung. Verwaltet die Tab-Leiste zur Steuerung der derzeitigen Ansicht.

`SettingsActivity` Die Einstellungsansicht, über die ein Graph geladen und gespeichert werden kann.

`MinimapActivity` Zeigt die Minimap, beziehungsweise die Übersicht des aktuell geladenen Graphen an. Bietet die Möglichkeit, einen Knoten auszuwählen.

`DetailNodeActivity` Zeigt den aktuell ausgewählten Knoten im Detail an, um ihn zu bearbeiten.

`Settings` Eine Singleton-Klasse mit Einstellungen zur zentralen Datenhaltung. Sie enthält alle Daten der aktuellen Sitzung.

`FlowDisplay` Die Anzeigefläche der Graphen-Übersicht. Unter Android wird diese als View implementiert und enthält einen Canvas zum Zeichnen des visuellen `AndroidGraph`. Hierbei werden die visuellen Knoten mit dem Präfix `Visual` aus `DBFilter.Android` verwendet, die sich selbstständig auf dem übergebenen Canvas zeichnen können.

Des Weiteren bindet das Projekt alle relevanten Ressourcen, darunter `String`-Ressourcen und `Drawable`-Ressourcen zur Darstellung der graphischen Elemente.

6.8 Benutzeroberfläche des Prototyps

Abbildungen 6.3, 6.4 und 6.5 zeigen die Benutzeroberfläche des fertig implementierten Prototyps, so wie er für die Evaluierung in Kapitel 7 verwendet wurde. Die schwarze Leiste am unteren Bildschirmrand stammt aus *Android 4.0* und stellt allgemein verfügbaren Bedienelemente dar. Hierzu gehören der Zurück-Button links, der Home-Button in der Mitte, der Task-Manager rechts, sowie ein kontextabhängiger Menü-Button am äußersten rechten Rand der Leiste. Darüber befindet sich die Tab-Leiste zur Auswahl der derzeitigen Ansicht.

Abbildung 6.3 zeigt die Startansicht des Prototyps. Hier kann ein Graph aus einer Liste aller verfügbaren Dateien geladen und unter einem beliebigen Namen gespeichert werden. Für die Evaluierung wurde das Textfeld zum Benennen der Datei ausgeblendet um das Speichern zu vereinfachen. Dateien wurden hierbei automatisch nach einem zeitbasierten Schema benannt.

Abbildung 6.4 zeigt die Übersichtsansicht eines Graphen. Das rote Auswahlrechteck markiert den derzeit im Detail sichtbaren „Genre“-Knoten. Wechselt der Benutzer in die Detailansicht, so sieht er den Knoten wie in Abbildung 6.5 dargestellt. Der Knoteninhalt ist blau dargestellt, anschließende Rezeptoren und Emitter grün. Im Knoten sind zwei Knoteneigenschaften dargestellt: Das zu filternde Datenbankattribut „Genre“, das über ein Textfeld editiert wird, sowie der gewählte Operator für die Vergleichsoperation, der über eine Gruppe aus Radio-Buttons festgelegt wird. Darunter befindet sich ein Button zum Markieren eines Knotens. Dieser wird lediglich für die durchzuführende Benutzerstudie verwendet. Am oberen Rand des Knoteninhalts, rechts vom beschriebenen Knotentyp, befindet sich der Button zum Entfernen des Knotens.

Abbildung 6.3: Benutzeroberfläche der Einstellungen

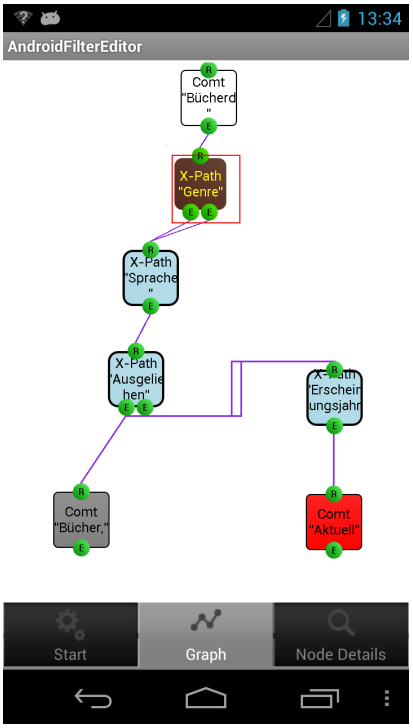


Abbildung 6.4: Benutzeroberfläche der Graph-Übersicht

Da bei diesem Knotentyp mehrere Emitter erlaubt sind, gibt es rechts neben diesen einen „+“-Button zum Hinzufügen eines neuen Emitters. Die Emitter selbst zeigen den zu filternden Wert an und können zur Bearbeitung angeklickt werden. Über dem Button mit der Beschriftung „New“ wird das Verbinden einer neuen Kante gestartet, wie es in Kapitel 5.3.10 beschrieben ist. Am oberen und unteren Rand sind die an den Konnektoren angeschlossenen Kanten dargestellt. Diese sind mit dem Knotentyp beschriftet, mit welchem sie verbunden sind.

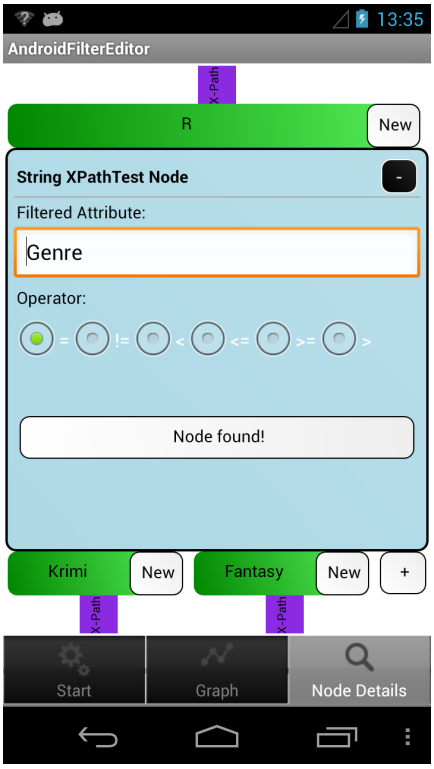


Abbildung 6.5: Benutzeroberfläche der Detailansicht

7 Evaluierung

In diesem Kapitel wird das entwickelte Konzept aus Kapitel 5 anhand einer Benutzerstudie evaluiert. Das Kapitel gliedert sich in vier Unterkapitel und beschreibt zunächst Methodik und Ablauf der Studie. Anschließend werden alle Ergebnisse, wie sie in der Studie aufgenommen wurden, aufbereitet dargestellt. Am Ende des Kapitels werden die erfassten Ergebnisse ausgewertet und diskutiert, um ein Fazit ziehen zu können.

7.1 Einleitung und Theorie

Gegenstand der Evaluierung ist das in Kapitel 5 entwickelte Konzept zur Darstellung von Filter/Flow-Graphen auf mobilen Geräten. Es wurde anhand des implementierten Prototyps aus Kapitel 6 getestet. Zusätzlich wurde ein Zoom-Mockup zum Vergleich eines traditionellen „Pan and Zoom“-Konzeptes untersucht. Es sollten qualitative Aspekte überprüft werden, um die Entwicklungsrichtung des Konzepts bestätigen oder korrigieren zu können. Die Benutzerstudie wurde am Institut für Visualisierung und interaktive Systeme durchgeführt. Dabei diente sie dazu, die aufgestellten Hypothesen aus Kapitel 5.2 belegen zu können. Des Weiteren sollte das Interaktionskonzept, wie es anhand der vorgestellten Use-Cases aus Kapitel 5.3 beschrieben ist, in der Praxis geprüft werden.

7.2 Methode

In diesem Abschnitt wird beschrieben, wie die Benutzerstudie durchgeführt wurde, welche Materialien zur Durchführung verwendet wurden und wer an der Studie teilgenommen hat.

Bei der Studie handelte es sich nach [18] um eine „Within-Subject“-Studie. Das bedeutet, dass es nur eine Gruppe von Probanden gab, die alle die gleichen Aufgaben durchgeführt haben. Des Weiteren beziehen sich die aufgestellten Hypothesen nicht auf bestimmte Probanden-Gruppen, sondern ausschließlich auf den zu evaluierenden Gegenstand.

7.2.1 Studienteilnehmer

Studienteilnehmer waren vorwiegend Studenten und Mitarbeiter der Universität Stuttgart, sowie anderer Hochschulen. Diese wurden mittels E-Mails oder Mund-Propaganda für

die Studie angeworben. Die Teilnehmer der Studie werden im Folgenden als Probanden bezeichnet. Es waren keine besonderen Vorkenntnisse zur Durchführung erforderlich. Die Probanden wurden nur statistisch erfasst. Es sollten mindestens zehn Probanden zur Durchführung der Aufgaben gefunden werden, die tatsächliche Teilnehmerzahl betrug elf.

Alle Probanden mussten zu Beginn einen Fragebogen ausfüllen, sowie einen Test auf Rot-Grün-Sehschwäche anhand von Ishihara-Farbtafeln durchführen. Das Alter aller Probanden betrug im Schnitt 25,7. Die Probanden waren durchweg männlich und Studenten an einer Hochschule. Somit waren die angestrebten Abschlüsse alle im Hochschulbereich angesiedelt, davon 72% Diplomstudenten. Die Computerkenntnisse wurden von 72% als „Fachmännisch“ eingestuft, die Übrigen stuften sie als „umfangreich“ ein.

Drei der elf Probanden waren Linkshänder, was sich besonders auf die Ergebnisse der Bedienung auswirkte. Eine Mehrheit von 7 Probanden trugen eine Sehhilfe. Weitere 3 Probanden hatten eine leichte Rot-Grün-Schwäche aufzuweisen. Wie sich herausstellen sollte, hatte dies jedoch keinen negativen Effekt auf die Bearbeitung der gestellten Aufgaben.

Die deutliche Mehrheit von 9 Probanden waren im Besitz eines Smartphones und mit dessen Bedienung vertraut. Lediglich 2 Probanden gaben an, kein Smartphone zu besitzen, hatten zuvor aber bereits eines bedient. Von den Probanden mit Smartphone nutzte jeweils genau eine Person das System „iOS“ sowie „Windows Phone 7“. Die restlichen sieben Probanden nutzten ein „Android“ System, wie es auch auf dem Testgerät installiert war. Somit hatte die Mehrheit aller Probanden Kenntnisse über die Bedienung des Betriebssystems und brachten Nutzungsgewohnheiten mit in die Studie. Die durchschnittlich angegebene Nutzungsdauer lag genau im Mittel von „Bis zu 10h“. Fünf Probanden gaben an, das Smartphone sowohl privat, als auch geschäftlich zu nutzen. Der Rest nutzt es ausschließlich privat. Somit gab es keinen Probanden, der sein Smartphone rein geschäftlich nutzte. Interessanterweise nutzten alle Smartphone-Besitzer ihr Gerät zum Informieren, wohingegen nur 88% angaben, es zum Kommunizieren zu nutzen. Man kann somit einen eindeutigen Trend des Smartphones als Informationsbeschaffungsmedium identifizieren. 77% nutzten ihr Smartphone allgemein zum Surfen im World Wide Web, Spielen gaben nur 55% als Nutzungsart an. Lediglich 22% gaben an für ihr Smartphone Apps zu programmieren. Von allen elf Probanden gaben zwei an, bereits Kenntnisse im Bereich Visualisierung vorweisen zu können.

7.2.2 Materialien

Die verwendeten Materialien dienten der Durchführung der Benutzerstudie und waren selbst nicht Gegenstand der Evaluierung.

- Ein Android-Smartphone vom Typ *Galaxy Nexus* zur Bearbeitung der Aufgaben. Die technischen Spezifikationen finden sich in Kapitel 3.3. Darauf installiert waren der Prototyp, zur Anzeige und Bedienung des entwickelten Konzeptes, sowie das Zoom-Mockup für Aufgabe 1.

- Ein Computer zur Präsentation des Tutorials und Auswertung digitaler Formulare.
- Materialien zur Durchführung der Testfälle, unter Anderem Beschreibung der notwendigen Arbeitsschritte in einem Tutorial. Das Tutorial wurde hierbei digital auf dem Präsentationscomputer zur Verfügung gestellt. Des Weiteren wurden Graphen-Abbildungen für die Orientierungsaufgaben der Hostapplikation als Bildmaterial zur Verfügung gestellt. Es wurde pro Orientierungsaufgabe ein Abbildungsblatt ausgegeben, insgesamt somit sechs Blätter. Die Probanden haben alle die gleichen Materialien zur Durchführung erhalten.
- Materialien zur Aufzeichnung der Ergebnisse. Hierzu gehörten der Fragebogen zur statistischen Erfassung des Probanden in einfacher Ausführung, der *NASA-TLX*-Fragebogen zur Bewertung der einzelnen Aufgaben in neunfacher Ausführung, sowie ein Auswertungsbogen für Freitext in einfacher Ausführung. Der Fragebogen diente zur Erfassung der folgenden statistischen Daten:
  - Alter
  - Geschlecht
  - Nutzung einer Sehhilfe
  - Rechts- oder Linkshänder
  - Computerkenntnisse (Gering | Umfangreich | Fachmännisch)
  - Besitz eines Smartphones
  - Betriebssystem des Smartphones (Android | iOS | Windows Phone 7)
  - Dauer der Smartphone-Nutzung pro Woche (5h | 10h | Mehr als 10h)
  - Art der Smartphone-Nutzung (Privat | Geschäftlich | Beides)
  - Angestrebter Abschluss (Bachelor | Master | Diplom | Doktor | Staatsexamen | Sonstige)
  - Kenntnisse von Visualisierungstechniken

7.2.3 Ablauf der Studie

Zu Beginn der Studie wurde den Probanden ein statistischer Fragebogen zum Ausfüllen ausgehändigt. Das Ausfüllen der Fragebögen im Voraus trug dazu bei, bei Fragen besser auf die Teilnehmer eingehen zu können, beispielsweise wenn keine Kenntnisse im Bereich der Smartphone-Nutzung vorlagen. Zudem wurde ein Ishihara-Farbtest zum Aufdecken einer Rot-Grün-Schwäche durchgeführt. Der Fragebogen befindet sich zur Einsicht im Anhang.

Die Probanden durften anschließend mit dem Tutorial beginnen. Dieses erklärte zu Beginn den Hintergrund der Studie und des entwickelten Prototyps. Es wurden Filter/Flow-Graphen anhand von Grafiken erklärt, um die Probanden auf die Aufgaben vorzubereiten.

Anschließend wurde das entwickelte Konzept vorgestellt. Fragen wurden vom Betreuer umgehend beantwortet.

Nach der Einführung wurde der erste Aufgabenblock von den Probanden durchgeführt. Jede Aufgabe wurde hierbei zunächst ausführlich erklärt. Bei der Umsetzung wurden den Probanden stets die einzelnen Schritte zur Durchführung der Aufgabe angezeigt. Sie konnten somit jederzeit nachsehen, ob sie richtig handelten. Nach der Durchführung jeder Aufgabe bekamen die Probanden einen *NASA-TLX*-Fragebogen ausgehändigt, den sie ausfüllen mussten.

Die Bearbeitung des zweiten Aufgabenblocks verlief analog zum Ersten. Am Ende beider Blöcke wurde den Probanden ein Freitext-Fragebogen ausgehändigt, den sie ausfüllen sollten. Anschließend wurde Feedback zur Studie und dem entwickelten Konzept innerhalb einer kurzen Diskussion eingeholt. Die Dauer der Studie betrug im Schnitt ungefähr eine Stunde. Die Studie wurde nach Abschluss mit 10 Euro vergütet.

Die Benutzerstudie wurde ohne Zeitvorgabe durchgeführt, um die Probanden nicht unter Druck zu setzen oder zu verunsichern. Die Probanden wurden darüber informiert, dass eine Zeitmessung zur Auswertung der Daten durch die Applikation erfolgte, aber sie die Aufgaben deshalb nicht hastig zu bearbeiten hatten. Bei Fragen und Problemen wurde jederzeit Hilfestellung geboten, sofern das Ergebnis der Aufgabe dadurch nicht verfälscht wurde. Die Rahmenbedingungen, wie der Ort und die verwendeten Materialien, waren für alle Probanden gleich.

7.2.4 Ort der Durchführung

Die Benutzerstudie wurde am Institut für Visualisierung und interaktive Systeme durchgeführt. Es wurde ein neutraler Raum zur Verfügung gestellt, um die ungestörte Durchführung der Studie zu gewährleisten. Das Institut stellte gleichzeitig alle zur Durchführung benötigten Utensilien, wie etwa das mobile Testgerät zur Ausführung des Prototyps, bereit.

7.3 Aufgaben

In diesem Abschnitt werden die von den Probanden durchgeführten Aufgaben beschrieben. Diese leiten sich aus den Hypothesen aus Kapitel 5.2 ab. Hierbei gliedern sich die Aufgaben in zwei Teilbereiche: Orientierung & Navigation, sowie Prüfung des Interaktionskonzepts.

7.3.1 Orientierung & Navigation

Bei diesem Aufgabengebiet sollen die Hypothesen zur Orientierung und Navigation mittels des entwickelten Konzeptes belegt werden. Die Probanden müssen hierbei Aufgaben erledigen, die das Auffinden von Knoten in Filter/Flow-Graphen auf dem Mobilgerät in

einem Zoom-Mockup, im Prototyp, sowie dem Originalgraphen der Referenzanwendung für den Desktop, beinhalten. Die folgenden vier Fälle wurden durch Aufgaben abgedeckt:

1. Auffinden von Knoten im Zoom-Mockup
2. Auffinden von Knoten im Prototyp
3. Auffinden von Knoten des Originalgraphen im Prototyp
4. Auffinden von Knoten des Prototyps im Originalgraph

Die Originalgraphen werden den Probanden hierbei als Abbildungen der Desktopanwendung präsentiert, um den Aufwand für die Studie beherrschbar zu halten. Durch die Punkte drei und vier soll gezeigt werden, dass das Konzept geeignet ist, die kollaborative Arbeit an Graphen mit Hilfe des Prototyps zu ermöglichen.

Zum Vergleich der Navigationsmöglichkeit wird ein Zoom-Mockup eines Filter/Flow-Graphen verwendet, welches das Prinzip aus 4.1.1 repräsentiert. Es soll gezeigt werden, dass das naive Darstellungskonzept mit Pan und Zoom für die Darstellung großer Graphen weniger geeignet ist, als das eigens entwickelte Konzept.

Für die Auswertung wurde die Bearbeitungsdauer und Korrektheit jeder Aufgabe gemessen und festgehalten.

Aufgabe 1: Auffinden von Knoten im Zoom-Mockup

Aufgabe 1 und 2 leiten sich von Hypothese zwei aus Kapitel 5.2 ab. Der Proband sollte einen bestimmten Filterknoten im Zoom-Mockup des Originalgraphens ausfindig machen. Hierzu musste er das Zoom-Mockup betrachten und den gewünschten Knoten anhand des gefilterten Attributes oder Emitterwerten ausfindig machen. Hatte er ihn gefunden, so musste er dies dem Betreuer mitteilen. Anschließend sollte die Anwendung vom Probanden beendet werden.

Der Proband musste die folgenden Schritte durchführen:

1. Starten der Mockup-Anwendung auf dem Smartphone
2. Auffinden des gesuchten Knotens
3. Mitteilen des gefundenen Knotens
4. Beenden der Anwendung mittels des „Zurück“-Buttons

Die Aufgabe wurde bei der Auswertung mit den Ergebnissen aus Aufgabe 2 verglichen. Beide Aufgaben waren in drei Unteraufgaben aufgeteilt, in denen zunächst ein Filterattribut gefunden werden musste, ein Wertepaar und schließlich ein Filterattribut abhängig vom Knoten-Kontext. Somit mussten die Probanden bei der letzten Aufgabe die Ergebnismengen betrachten, die semantisch mit dem gesuchten Knoten verknüpft waren.

Die Zeit wurde ab dem Start der Applikation, bis zum Abschluss der Aufgabe durch Beenden der Applikation, festgehalten und in einer Datei gespeichert. Die Korrektheit der gefundenen Knoten wurde durch den Betreuer der Studie überprüft und festgehalten.

Aufgabe 2: Auffinden von Knoten im Prototyp

Der Proband sollte einen bestimmten Filterknoten im Prototyp ausfindig machen. Hierzu musste er einen vorgegebenen Graphen laden und nach dem gegebenen Knotentypen suchen. Der zu suchende Knoten wurde hierbei entweder über das zu filternde Attribut, oder seine möglichen Filterwerte in den Emittern, definiert. Hatte der Proband ihn gefunden, so musste er in die Detailansicht des Knotens wechseln und ihn über einen Knopf markieren. Die Aufgabe diente, unter Anderem, dem Vertrautmachen mit dem Overview-Detail-Konzept und der Navigation zu gesuchten Knoten. Spätere Aufgaben bauten auf diesem Konzept auf.

Zur Durchführung waren die folgenden Schritte notwendig:

1. Laden des vorgegebenen Graphen
2. Navigation zum gesuchten Knoten in der Übersicht
3. Wechsel in die Detailansicht
4. Drücken eines vordefinierten Buttons zur Markierung des Knotens
5. Speichern des Graphen

Die Zeit wurde ab dem Ladevorgang des Graphen, bis zum Abschluss der Aufgabe durch Drücken des Knopfes, festgehalten. Diese wurde, zusätzlich zum Graph, in der Datei gespeichert. Durch das Speichern des über den Knopf markierten Knotens konnte anschließend ausgewertet werden, ob der richtige Knoten gefunden wurde.

Aufgabe 3: Auffinden von Knoten des Originalgraphen im Prototyp

Aufgaben 3 und 4 leiten sich von Hypothese drei aus Kapitel 5.2 ab. Es sollte gezeigt werden, dass die Probanden keine Probleme bei der Übertragung von einer Visualisierung auf die jeweils andere haben und sich das Minimap-Konzept zur kooperativen Arbeit mit einer Host-Anwendung eignet. Bei Aufgabe 3 sollte die Übertragung von Informationen vom Originalgraphen auf den Prototyp untersucht werden, während Aufgabe 4 sich mit der Rückrichtung beschäftigt.

Der Proband sollte einen bestimmten Filterknoten aus der Abbildung des Originalgraphen im Prototyp ausfindig machen. Hierzu musste er den Originalgraphen betrachten und sich den markierten Knoten merken. Nun sollte er den markierten Knoten auf dem Mobilgerät finden. Hatte er ihn gefunden, so musste er ihn mit Hilfe eines Knopfes markieren. Hierzu musste der Proband in die Detailansicht wechseln. Die Aufgabe zielte auf die Orientierung beim Wechsel zwischen den Darstellungen von der Hostanwendung

auf die Mobilanwendung ab. Sie sollte zeigen, dass dem Proband ein Informationstransfer möglich ist.

Zur Durchführung waren folgende Schritte notwendig:

1. Laden des Graphen im Prototyp
2. Umdrehen des erhaltenen Blattes mit der Abbildung des Originalgraphen
3. Merken des markierten Knotens anhand Position, Farbe und Attributen
4. Wechsel zum Prototyp
5. Auffinden des gesuchten Knotens in der Strukturansicht des Prototyps
6. Auswahl des Knotens und Wechsel in die Detailansicht des Knotens
7. Markierung des gesuchten Knotens als gefunden durch Drücken des Knopfes
8. Wechsel zur Startansicht und Speichern des Graphen

Die Zeit wurde ab dem Laden des Graphen auf dem Mobilgerät, bis zum Abschluss der Aufgabe durch Drücken des Knopfes, festgehalten. Durch das Markieren über den vorgegebenen Knopf konnte anschließend ausgewertet werden, ob die Bearbeitung korrekt erfolgt ist.

Aufgabe 4: Auffinden von Knoten des Prototyps im Originalgraphen

Aufgabe 4 beschäftigt sich mit der Übertragung von Informationen vom Prototyp auf den Originalgraphen der Host-Anwendung. Der Proband sollte einen bestimmten Filterknoten aus dem Prototyp im Originalgraphen ausfindig machen. Hierzu musste er den Graph auf dem Mobilgerät laden, ihn betrachten und sich den markierten Knoten merken. Hierfür wurde die Mobilapplikation so modifiziert, dass sie den zu suchenden Knoten visuell hervorhob. Nun sollte er den markierten Knoten in der Abbildung des Originalgraphen finden. Hatte er ihn gefunden, so sollte er ihn markieren. Die Aufgabe zielte auf die Orientierung beim Wechsel zwischen den Darstellungen von der Mobilanwendung auf die Hostanwendung ab. Sie sollte zeigen, dass dem Proband ein Informationstransfer möglich ist.

Die folgenden Schritte waren notwendg:

1. Laden des Graphen im Prototyp
2. Merken des markierten Knotens anhand Position, Farbe und Attributen
3. Auffinden des gesuchten Knotens in der Abbildung des Originalgraphen
4. Markieren des Knotens in der Abbildung des Originalgraphen
5. Speichern des Graphen auf dem Mobilgerät

Die Zeit wurde ab dem Ladevorgang, bis zum Abschluss der Aufgabe durch Speichern des Graphen, festgehalten. Durch das Markieren konnte anschließend ausgewertet werden, ob die Bearbeitung korrekt erfolgt ist.

7.3.2 Prüfung des Interaktionskonzepts

Die folgenden Aufgaben leiten sich von Hypothese vier und fünf aus Kapitel 5.2 ab und sollten zeigen, dass das entwickelte Konzept geeignet ist, die beschriebenen Anforderungen aus Kapitel 3.4 zu erfüllen. Hierfür wurden Interaktionen, die in Kapitel 5.3 beschrieben sind, anhand des Prototyps überprüft. Bei der Prüfung des Interaktionskonzepts sollten die folgenden Fälle durch Aufgaben abgedeckt werden:

1. Bearbeiten vorhandener Knoten, 5.3.4
2. Hinzufügen von Knoten, 5.3.5
3. Löschen von Knoten, 5.3.6
4. Löschen von Konnektoren, 5.3.8
5. Hinzufügen von Konnektoren, 5.3.7
6. Hinzufügen einer Kante, 5.3.10

Die Auswahl erfolgte so, dass jede Interaktionsklasse abgedeckt wurde. Es wurde beispielsweise das Löschen von Kanten nicht zusätzlich geprüft, da das Berühren eines Elementes zum Löschen durch die restlichen Aufgaben bereits abgedeckt wird.

Zur Durchführung der Aufgaben wurden klare Aufgabenbeschreibungen vorgegeben, die in den folgenden Kapiteln erklärt sind. Für die Auswertung wurde die Bearbeitungsdauer und Korrektheit jeder Aufgabe gemessen und festgehalten. Die Zeit wurde ab dem Ladevorgang des Graphen, bis zum Speichervorgang nach Abschluss der Aufgabe, festgehalten. Diese wurde, zusätzlich zum Graph, in der Datei gespeichert. Durch das Speichern des Graphen konnte anschließend ebenso ausgewertet werden, ob die Bearbeitung korrekt erfolgte.

Bei der folgenden Beschreibung der Aufgaben wird nur der allgemeine Ablauf beschrieben; die detaillierten Schritte können den Use-Cases oder dem Aufgaben-Tutorial entnommen werden.

Aufgabe 5: Bearbeiten vorhandener Knoten

Aufgabe 5 befasste sich mit der Navigation im Graphen und dem Verändern von Filterwerten. Die Aufgabe soll die erste Interaktion aus Kapitel 7.3.2 abdecken. Der Proband sollte vorgegebene Attribut- und Emitterwerte existierender Knoten ändern. Er musste hierzu den Knoten anhand der gegebenen Attribute und Werte identifizieren und auswählen. Anschließend musste er in die Detailansicht des Knotens wechseln und die geforderten Werte ändern. Danach sollte er den Graph speichern.

1. Laden des vorgegebenen Graphen
2. Auffinden des gesuchten Knotens anhand der gegebenen Werte

3. Wechsel zur Detailansicht
4. Auffinden des gesuchten Attributs
5. Änderung des gesuchten Attributs mit den gegebenen Werten
6. Speichern des Graphen

Aufgabe 6: Hinzufügen von Knoten

Aufgabe 6 deckt die zweite Interaktion aus Kapitel 7.3.2 ab. Der Proband sollte einen neuen Knoten eines bestimmten Typs an einer freien Stelle hinzufügen. Hierzu musste er eine freie Stelle in der Übersicht des Graphen mindestens 2 Sekunden lang berühren, um die Interaktion zu starten. Anschließend sollte er in der erschienenen Liste einen vorgegebenen Knotentyp auswählen und diesen einfügen. Mit Hilfe der Detailansicht sollte er dann noch das Attribut des Knotens initialisieren. Die Aufgabe wurde mit verschiedenen Knotentypen durchgeführt. Nach jeder Einfüge-Operation sollte der Proband den Graph speichern.

1. Laden des vorgegebenen Graphen
2. Langes Drücken an einer beliebigen freien Stelle in der Graphenübersicht
3. Auswahl des vorgegebenen Knotentyps
4. Bestätigung durch Drücken auf „Create“ zum Erstellen des Knotens
5. Initialisieren des Knotens durch setzen der vorgegebenen Attribute
6. Speichern des Graphen

Aufgabe 7: Löschen von Knoten und Emittern

Aufgabe 7 deckt die dritte und vierte Interaktion aus Kapitel 7.3.2 ab, das Löschen von Knoten und Emittern. Die Interaktionen wurden in dieser Aufgabe zusammengeführt, da die Aktion bei beiden Objekttypen gleich abläuft. Der Proband musste das gewünschte Objekt in der Detailansicht auswählen und einen Knopf zum Entfernen betätigen. Anschließend sollte der Graph gespeichert werden.

1. Laden des vorgegebenen Graphen
2. Auswahl des angegebenen Knotens
3. Wechsel in die Detailansicht des Knotens
4. Löschen des gegebenen Knotens oder Emitters durch Drücken auf den Entfernen-Knopf
5. Speichern des Graphen

Aufgabe 8: Hinzufügen von Emittern

Aufgabe 8 prüft die fünfte Interaktion aus Kapitel 7.3.2. Der Proband sollte eine neue Ergebnismenge hinzufügen, indem er einen neuen Emitter zu einem vorgegebenen Knoten hinzufügt. Hierzu musste er gegebenen Knoten auswählen und in dessen Detailansicht wechseln. Anschließend sollte er den Knopf zum Hinzufügen eines Emitters betätigen und den Emitter mit den vorgegebenen Werten initialisieren. Die Aufgabe wurde mit unterschiedlichen Emittertypen durchgeführt.

1. Laden des vorgegebenen Graphen
2. Auswählen des gesuchten Knotens
3. Wechsel in die Detailansicht des Knotens
4. Hinzufügen eines Emitters im Knoten durch Drücken des Knopfes
5. Initialisieren des Emitters mit den vorgegebenen Werten
6. Speichern des Graphen

Aufgabe 9: Hinzufügen von Kanten

In der letzten Aufgabe wurde Interaktion sechs aus Kapitel 7.3.2 abgedeckt. Hierbei ging es um das Hinzufügen von Kanten mittels des vorgestellten Copy & Paste-Prinzips. Der Proband musste hierzu den Startknoten in der Übersicht auswählen und in dessen Detailansicht wechseln. Hier musste er am vorgegebenen Konnektor auf den Knopf zum Verbinden klicken. Anschließend musste der Zielknoten in der Übersicht ausgewählt werden. In der Detailansicht des Zielknotens sollte nun die Kante, über einen Knopf am Zielkonnektor, angeschlossen werden.

1. Laden des vorgegebenen Graphen
2. Auswählen des Startknotens
3. Wechsel in die Detailansicht des Startknotens
4. Merken des Konnektors durch Drücken auf „New“
5. Auswahl des Zielknotens in der Graphenübersicht
6. Verbinden der Knoten durch Auswahl von „Con“ am Zielkonnektor
7. Speichern des Graphen

7.4 Ergebnisse

Im Folgenden werden alle während der Benutzerstudie erhobenen Messdaten festgehalten und übersichtlich dargestellt. Insgesamt wurden während der Studie die Ergebnisse von

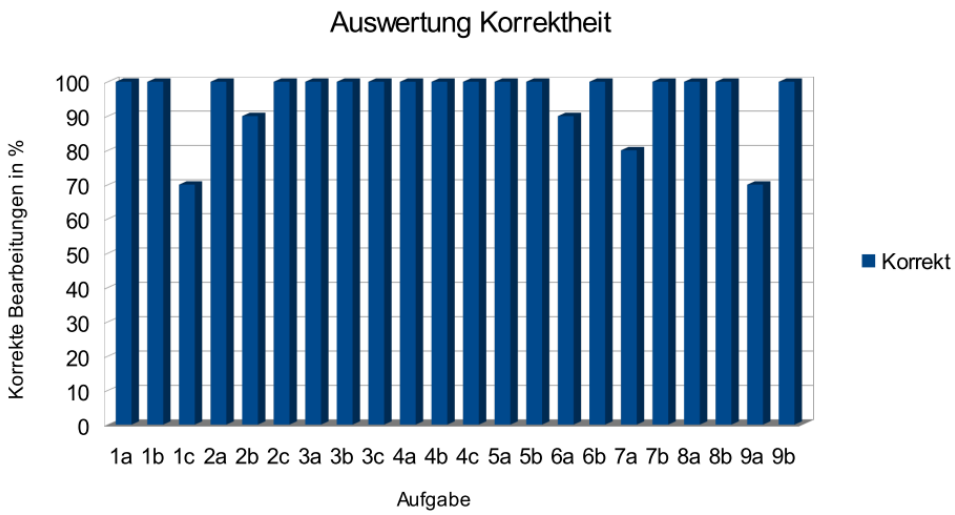


Abbildung 7.1: Anzahl korrekt bearbeiteter Aufgaben

elf Probanden aufgenommen. Hierbei wurden Korrektheit der bearbeiteten Aufgaben, die Dauer der Bearbeitung, sowie ein *NASA-TLX*-Fragebogen erfasst. Die Daten werden ohne Wertung abgebildet; eine Interpretation findet sich in Kapitel 7.5. Eine Auswertung der Freitexte und der Diskussion findet sich ebenso in Kapitel 7.5.

7.4.1 Korrektheit

Die Bearbeitungskorrektheit der Aufgaben wurden anhand von Speicherabbildern der jeweiligen Aufgabe geprüft. Lediglich die Antworten der ersten Aufgabe wurden auf Grund technischer Einschränkungen bereits während der Durchführung der Studie erfasst. Eine Aufgabe wurde dann als korrekt gewertet, wenn der Proband die Arbeitsschritte korrekt durchgeführt hat und das Ergebnis dem erwarteten Graphenabbild entspricht. Bei einer Bearbeitungsaufgabe mussten beispielsweise die richtigen Attribute im richtigen Knoten nach den vorgegebenen Werten verändert werden. Abbildung 7.1 gibt die Anzahl aller korrekt bearbeiteter Aufgaben in Prozent an. Die Werte wurden pro bearbeiteter Teilaufgabe erfasst. Die Y-Achse spiegelt hierbei die Anzahl an Probanden wider, die die Aufgaben korrekt gelöst haben.

Betrachtet man Abbildung 7.1 so sieht man, dass die Aufgaben drei, vier und fünf von allen Probanden korrekt gelöst wurden. Bei den Aufgaben zwei, sechs und sieben konnten Ausreißer festgestellt werden, die sich jedoch im statistischen Toleranzbereich bewegen. Aufgaben 1c und 9a wurden von nur 70% der Probanden korrekt bearbeitet. Die Gründe hierfür werden in Kapitel 7.5 genauer untersucht.

Insgesamt lässt sich erkennen, dass die Mehrheit der Aufgaben von allen Probanden korrekt bearbeitet wurde. Im Durchschnitt wurden 95% aller Aufgaben korrekt bearbeitet.

7.4.2 Bearbeitungsdauer

Die Dauer der Aufgabenbearbeitung wurde bei allen Aufgaben innerhalb des Programmes gemessen und in den Speicherabbildern festgehalten. Die Messgenauigkeit erfolgte somit in Millisekunden, wobei für die vorliegende Auswertung eine Genauigkeit im Sekundenbereich ausreichend ist, da kleinere Schwankungen im Toleranzbereich liegen. Bei der ersten Aufgabe wurde die Zeit von Start bis Beendigung der Applikation gemessen, bei den Aufgaben zwei und drei vom Laden des Graphen bis zum Drücken des „Node Found"-Buttons, bei allen weiteren Aufgaben vom Laden des Graphen bis zum Zeitpunkt des Speicherns. Abbildung 7.2 gibt den Mittelwert und den Median der Bearbeitungszeiten jeder Aufgabe wieder.

Bei der Betrachtung der Zeiten kann man beim Wechseln vom ersten zum zweiten Aufgabenblock, zwischen Aufgabe 4c und 5a, einen sprunghaften Anstieg feststellen. Interaktionsaufgaben haben somit generell länger gedauert, als reine Orientierungs- und Navigationsaufgaben. Im ersten Block können Spitzen bei den Aufgaben 1c und 2c festgestellt werden. Aufgaben 3 und 4 lieferten gleichmäßig verteilte Bearbeitungszeiten, wobei Aufgabe 4 schneller durchgeführt wurde als Aufgabe 3.

Im zweiten Aufgabenblock ab Aufgabe 5a können stärkere Fluktuationen bezüglich der Bearbeitungsdauer festgestellt werden. Vor allem die Aufgaben 6a und 9a stellen mit einer durchschnittlichen Dauer von über einer Minute Ausreißer dar. Gleichzeitig stellen die Dauer der Aufgaben 6b und 8b Minima bezüglich des Aufgabenblocks dar. Tabelle 9.4 im Anhang zeigt alle in der Studie erfassten Werte.

7.4.3 Nutzerbewertung

Um Tendenzen bezüglich der gefühlten Anstrengung der Probanden bei der Benutzung der Visualisierungen erkennen zu können, wurde der *NASA-TLX*-Fragebogen, zur Ermittlung des Task Load Index, verwendet. Hierbei wurden die folgenden Faktoren erfasst:

- Mentale Beanspruchung
- Physische Beanspruchung
- Kurzzeitige Beanspruchung
- Eigene Leistung/Bewertung
- Aufwand
- Frustration

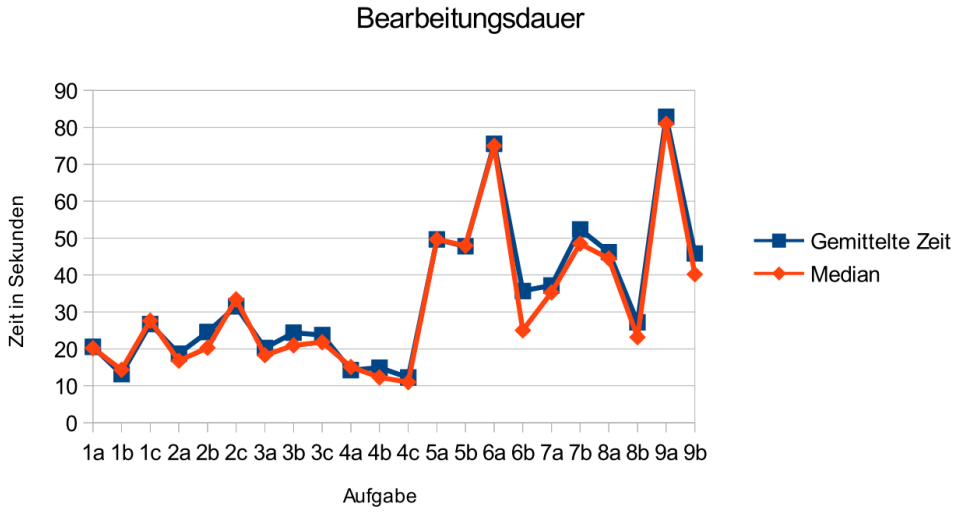


Abbildung 7.2: Bearbeitungsdauer der Aufgaben in Sekunden

Bei der vorliegenden Studie wurden die Faktoren nach jeder Aufgabe anhand einer 20-stufigen Skala vom Probanden bewertet. Durch die Berechnung von Mittelwert und Median der Ergebnisse konnten so Tendenzen festgestellt werden, die für oder gegen die aufgestellten Hypothesen sprechen. Es konnte beispielsweise die Beanspruchung bei Durchführung einer Interaktion gemessen werden, um zu ermitteln, ob die gewählte Interaktion angemessen ist.

In Abbildung 7.3 ist die Auswertung der Fragebögen für den jeweiligen *NASA-TLX*-Faktor aufgetragen. Hierbei wurde für jede Aufgabe der Mittelwert und der Median in die Diagramme eingetragen. Die verwendete Skala reicht von 0 bis 20, wobei Werte kleiner 10 als gering und Werte größer 10 als hoch eingestuft werden. Größere Werte stellen somit eine schlechtere Bewertung dar, da die Beanspruchung der Probanden höher war.

Bei der Auswertung des *NASA-TLX* musste ein Proband aufgrund mangelndem Interesse bei der Ausfüllung des Fragebogens aus der Auswertung gestrichen werden, da der Datensatz die Ergebnisse einseitig verfälscht hätte.

Mental Demand

Abbildung eins aus 7.3 zeigt die geistige Beanspruchung der Probanden verteilt über die Aufgaben. Die Beanspruchung über alle Aufgaben Betrug im Mittel 6,88. Die Abbildung lässt bei Aufgabe 4 mit einem Mittelwert von 3,2 und einem Median von 2,5 einen deutlichen Einbruch erkennen. Die Beanspruchung stieg daraufhin beim Wechsel zum Interaktionsblock mit Aufgabe 5 stark an. Der Mittelwert stieg auf 9, der Median auf 10

an. Aufgabe 9 wurde mit einem Mittelwert von 9,2 und einem Median von 9 als ebenso anspruchsvoll empfunden.

Physical Demand

Die zweite Abbildung aus 7.3 zeigt die körperliche Beanspruchung der Probanden bei der Bearbeitung der Aufgaben. Die Auswertung zeigt, dass die körperliche Beanspruchung durchweg gering war. Der Durchschnitt aller Aufgaben liegt bei 3,91, ohne Ausreißer. Einzig die Tendenzen für Aufgabe eins bis vier aus Abschnitt 7.4.3 lassen sich leicht wiedererkennen. Der Kennwert „Physical Demand“ war somit letztendlich ungeeignet für die vorliegenden Aufgabentypen und wird in der Auswertung nicht weiter betrachtet.

Temporal Demand

Abbildung drei aus 7.3 stellt den gefühlten Zeitdruck dar, dem die Probanden ausgesetzt waren. Die Kurve entspricht einer geschwächten Version aus Abschnitt 7.4.3. Der Durchschnitt aller Aufgaben lag mit 5,42 im unteren Bereich der Skala, die Probanden standen unter leichtem Zeitdruck. Aufgabe 4 stellt mit einem Mittelwert von 4,1 den besten Wert des ersten Blockes, Aufgabe 6 mit 5 den besten Wert des zweiten Blockes dar. Aufgabe 5 sticht als erste Aufgabe des zweiten Blockes erneut hervor mit einem Höchstwert von 7,2 für den Mittelwert und einem Median von 7,5.

Performance

Abbildung vier aus 7.3 gibt die Einschätzung der Probanden ihrer eigenen Leistung wieder. Die Probanden schätzten ihre Leistung mit 3,32 durchweg als sehr gut ein. Die Probanden waren somit sehr zuversichtlich, was die Umsetzung der ihnen gestellten Aufgaben betraf. Allerdings bringen die Ergebnisse kaum statistisch relevante Ausprägungen hervor. Die geschätzte Leistung für Aufgabe 1 und 4 weisen mit sehr niedrigen Werten von 1,7 und 2,8 die Tendenz auf, dass die Aufgaben am besten bearbeitet worden sind. Dies deckt sich mit den Tendenzen der anderen Faktoren.

Effort

Die fünfte Abbildung aus 7.3 zeigt den benötigten Aufwand für die Bearbeitung der jeweiligen Aufgabe. Die Bewertung des Aufwandes zeigt sich im Vergleich mit den anderen Faktoren wieder als recht heterogen. Wie bereits in Abschnitt 7.4.3 wurden Aufgabe 4 und 6 für den jeweiligen Block am besten bewertet. Der Mittelwert von Aufgabe 4 liegt bei 3,7, der Median sogar nur bei 3. Die Kurve zeigt große Ähnlichkeit zu den Werten aus Abschnitt 7.4.3 und bekräftigt die Tendenzen für Aufgabe 5 und Aufgabe 9. Beide Aufgaben schneiden mit Mittelwerten von 8 und 9,3 bezüglich des Aufwandes am schlechtesten ab. Der Gesamtaufwand beträgt im Schnitt 6,63.

Frustration

Die letzte Abbildung aus 7.3 zeigt die gefühlte Frustration bei der Bearbeitung der jeweiligen Aufgabe. Die ermittelte durchschnittliche Frustration liegt bei 5,01. Allerdings ergeben sich bei Aufgabe 5 und 9, ähnlich zu den Werten aus Abschnitt 7.4.3 und Abschnitt 7.4.3, Spitzen mit 6,9 und 6,5. Kapitel 7.5 untersucht, ob die Aufgaben tatsächlich Probleme bereitet haben. Aufgabe 4 wurde mit 1,9 erneut am besten eingestuft.

Die Auswertung des NASA-TLX ergab, dass die geistige Anstrengung mit 6,5 für Aufgabe 3 und 3,2 für Aufgabe 4 als gering einzustufen ist und die Übertragung vom Smartphone auf die Host-Visualisierung einfacher war als umgekehrt. Mit dem Wert 3,2 ist Aufgabe 4 somit auch die einfachste Aufgabe, was die geistige Anstrengung betrifft. Die körperliche Anstrengung liegt mit 3,6 und 2,9 ebenfalls unter dem Durchschnitt. Dieser Trend setzt sich bei allen weiteren Kennwerten fort - die Kennwerte für Aufgabe 4 sind von allen Aufgaben am niedrigsten. Die Probanden schätzten ihre Leistung bei Aufgabe 4 mit 1,7 als perfekt ein, was auch zutraf, bei Aufgabe 3 immerhin als gut mit 4,6. Auch Aufwand und Frustration lagen mit 3,7 und 1,9 deutlich unter dem Durchschnitt, bei Aufgabe 3 mit 7,1 und 5,6 im Mittel.

7.5 Auswertung

Im Folgenden werden die erfassten Werte zueinander in Bezug gesetzt und interpretiert. Aussagen stützen sich zusätzlich auf die Antworten der Probanden im Fragebogen sowie der geführten Diskussionen.

7.5.1 Auswertung Navigation

Die Auswertung zur Navigation im Prototyp bezieht sich auf den in Aufgaben eins (Mockup) und zwei (Prototyp) geführten Vergleich bei der Suche nach Knoten. Nach Kapitel 7.4.1 gab es für die erste Navigationsaufgabe eine Korrektheitsverteilung von 100/100/70 Prozent. Diese erklärt sich damit, dass einige Probanden die semantische Abhängigkeit aus Aufgabe 1c ignoriert haben und einen ähnlichen Knoten einer falschen Ergebnismenge ausgewählt haben. Aufgabe zwei ergab eine Korrektheitsverteilung von 100/90/100 Prozent. Dies spricht für eine bessere Übersicht bei der Bearbeitung der letzten, semantischen Aufgabestellung, da der richtige Knoten leichter identifiziert werden konnte. Die Abweichung aus Aufgabe 2b lässt sich nur als Ausreißer interpretieren. Bei der Bearbeitungsdauer haben die Probanden im Schnitt fünf Sekunden länger für die Durchführung von Aufgabe zwei benötigt. Der zeitliche Unterschied ist somit gering und kann vom neuartigen Konzept herrühren.

Um die Konzepte weiter zu vergleichen, wird die Auswertung des NASA-TLX betrachtet. Hier lag die geistige Beanspruchung mit 7,4 gegenüber 5,4 bei dem Minimap-Konzept leicht höher als der naiven Implementierung mittels Zoom. Der gefühlte Zeitdruck war

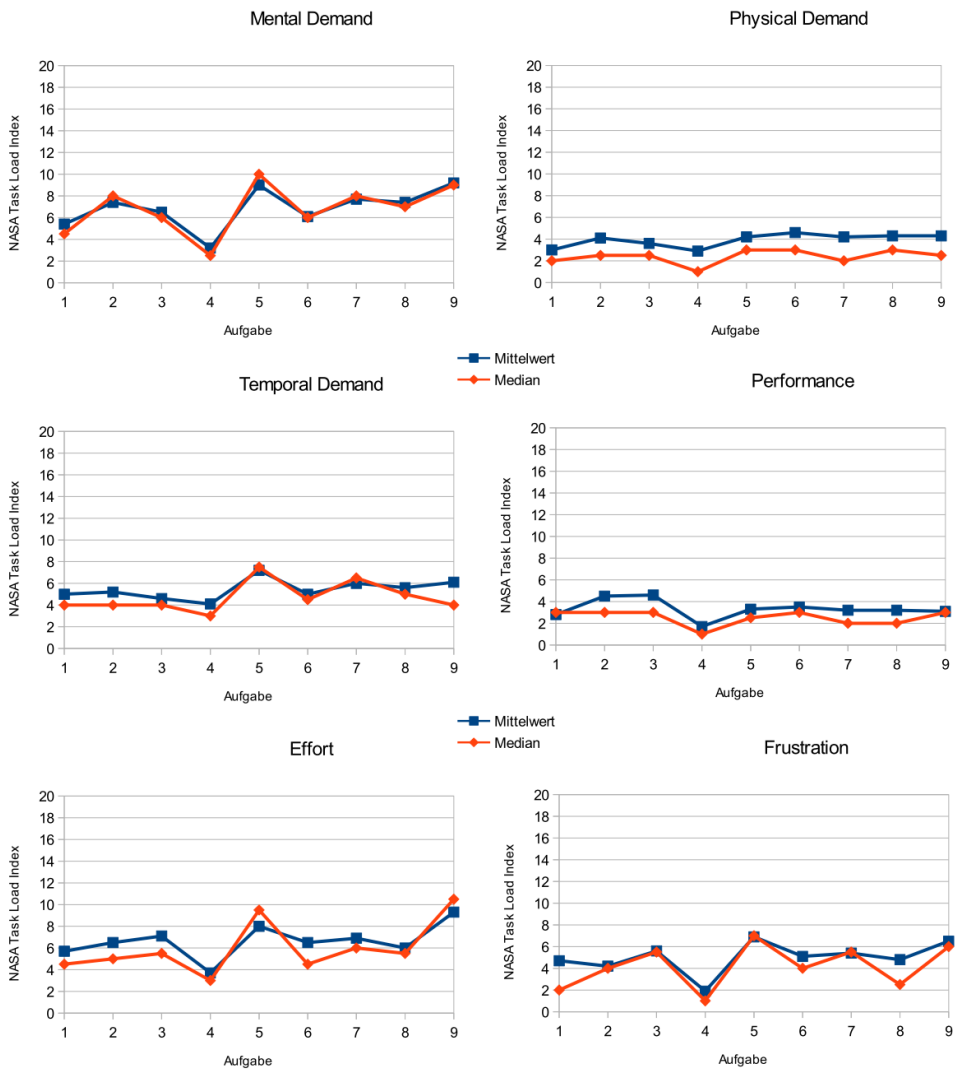


Abbildung 7.3: Auswertung der sechs Faktoren des NASA-TLX für zehn der elf Probanden.

mit 5,2 gegenüber 5,0 nahezu gleich. Die Probanden schätzten, entgegen der besseren Auswertung bei der Korrektheit, ihre Leistung mit 4,5 gegenüber 2,8 bei der Nutzung des Minimap-Konzepts schlechter ein. Dies könnte von der Nutzung dieses neuartigen Konzeptes herrühren, das die Probanden verunsichert hat. Ebenso wurde der Aufwand mit 6,5 gegenüber 5,7 leicht höher eingestuft, was vermutlich von der Gewöhnung an das Bedien-Prinzip kommt. Interessanterweise wurde die gefühlte Frustration mit 4,2 gegenüber 4,7 geringer eingestuft. Dies spricht für die oben aufgeführte Hypothese.

Eine Auswertung bezüglich der Übersicht im Prototyp ergab, dass 90 Prozent aller Probanden die Übersicht bei der Navigation gut fanden und die Knoten besser lesbar seien, als im Zoom-Konzept. Angemerkt wurden vorwiegend das Bedienkonzept mittels Auswahlrechteck, sowie kleinere visuelle Aspekte, wie die Textlänge innerhalb der Knoten, bei der ein Proband kritisierte, dass dieser am Rand nicht vollständig lesbar sei. Dies kann durch einen Zwischenraum jedoch behoben werden. Das Bedienkonzept wird in Kapitel 7.5.2 diskutiert.

Die Implementierung eines Zoom-Konzeptes befürworteten 60 Prozent, 30 Prozent entschieden sich dagegen, der Rest enthielt sich einer Meinung. Von den Befürwortern äußerte sich ein Proband, dass man „zwar eine Touch-Auswahl implementieren, aber das Zoom-Konzept beibehalten solle“. Somit stellte sich heraus, dass er die Detailansicht missverständlich als Zoom interpretiert hat. Ein weiterer Proband gab an, dass er auf keinen Fall eine Zoom-Ansicht haben wollte, da er dies bereits bei der Bedienung eines Browsers sehr umständlich findet. Einige andere Probanden meinten, das vorgestellte Zoom-Mockup sei für die verwendete Graphengröße in Ordnung gewesen, würde aber bei größeren Graphen sicherlich zu Problemen führen. Bereits bei der getesteten Graphengröße begannen die Probanden großteils damit, eine Zoom-Operation auszuführen, um die Knoten lesen zu können, bevor sie überhaupt die Graphstruktur betrachteten.

Für eine Zoomfunktion argumentierte ein Proband, die Textgröße der Minimap sei zwar noch lesbar, aber bereits an der Grenze. Er gab an, dass sein Vater mit Lesebrille Probleme haben würde, die Texte lesen zu können. Er meinte gleichzeitig, dass es jedoch kein Widerspruch zu der vorgestellten Trennung in Detailansicht zur Bearbeitung und der Minimap sei und man die Zoom-Funktion zur Lesbarkeit in die Minimap einarbeiten sollte. Dies wird dadurch bekräftigt, dass 90 Prozent der Probanden angaben, dass sie das Konzept der Trennung in eine Übersicht und eine Ansicht zum Bearbeiten gut fanden. Es könnte also ein Kompromiss gefunden werden, bei dem eine Zoom-Funktion zur besseren Lesbarkeit in die Minimap eingebaut wird, ohne diese vorrangig zur Navigation nutzen zu müssen.

Um signifikante Ergebnisse beim Konzeptvergleich zu erzielen, müsste das Konzept vollständig implementiert werden und mit sehr großen Graphen erneut gegen das naiv implementierte Mockup antreten. Man kann jedoch festhalten, dass der Prototyp eine gute Übersicht bietet und semantische Abhängigkeiten leichter erkennen lässt.

7.5.2 Auswertung Orientierung

Die folgende Auswertung bezieht sich auf die Aufgaben drei und vier, zur Orientierung zwischen Prototyp und Originalgraph. Nach 7.4.1 konnten die Orientierungsaufgaben von allen Probanden korrekt gelöst werden. Die Auswertung der Bearbeitungsdauer ergab, dass die Aufgaben in durchschnittlich 20 Sekunden gelöst und somit relativ schnell bearbeitet wurden. Aufgabe vier schnitt durchweg 10 Sekunden besser ab als Aufgabe drei. Die Auswertung des NASA-TLX bestätigt diesen Trend – Aufgabe vier schneidet sowohl bezogen auf den Aufgabenblock, als auch auf die gesamten Aufgaben am besten ab und gilt damit als leichteste Aufgabe. Die Übertragung vom Speziellen, hier dem Prototyp, auf das große Original der Desktopapplikation viel den Probanden somit leichter. Dies ist auch damit zu erklären, dass bei Aufgabe 3 mehr Interaktion vom Benutzer gefordert wurde, in Form von Ansichtwechsel und Interaktion mit einem Bedienelement. Bei Aufgabe 4 musste der gefundene Knoten lediglich per Hand markiert werden, da die Hostapplikation nicht für den Test modifiziert wurde. Das Overview-Detail-Konzept stellt somit kein Hindernis bei der Zusammenarbeit mit dem Originalkonzept der Host-Applikation dar.

Kleinere Probleme gab es lediglich bei Teilaufgabe 4c. Es galt einen Knoten vom Typ Textvergleich zu finden. Ein Proband gab an, dass er bei den Vergleichsaufgaben nach Details gesucht hat, die auf dem Smartphone nicht auf den ersten Blick vorhanden waren. Dies kann damit begründet werden, dass die Minimap aus Platzgründen keine Emitterwerte in den Knoten darstellen kann. Um diese zu prüfen, muss der Benutzer vorher in die Detailansicht wechseln, was ein fester Bestandteil des Konzeptes ist. Dieser Umstand sorgte bei dieser Teilaufgabe für eine Erhöhung der Bearbeitungsdauer, wirkte sich jedoch nicht negativ auf die Korrektheit der Antworten aus.

Bekräftigt wurde die positive Auswertung durch die Aussagen der Probanden. Nahezu alle Probanden bewerteten den Überblick im Prototyp als gut, siehe Kapitel 7.5.1, was die leichte Orientierung begünstigt. Ein Proband gab zusätzlich an, dass er den Überblick in der Minimap gut fand, da die Anordnung die gleiche sei, wie im Original. Der Erhalt der geometrischen Struktur spielt somit eine wichtige Rolle für den Wechsel der Ansichten. Zusammenfassend lässt sich sagen, dass es keine Probleme bei der Orientierung zwischen den verwendeten Visualisierungen gab.

Auswertung der Interaktionskonzepte

Die folgende Auswertung bezieht sich auf den Interaktionsblock mit den Aufgaben fünf bis neun. Bezogen auf alle Interaktionen waren Aufgaben 5 und 9 offensichtlich am schwierigsten umzusetzen. Die Bearbeitungsdauer war bei ihnen am höchsten, dies spiegelt sich auch in den Werten für die geistige Anstrengung und den Aufwand aus dem NASA-TLX wider. Entgegen dessen waren die Aufgaben 6 und 8 offensichtlich am einfachsten durchzuführen, was an den Senken in den Diagrammen abzulesen ist.

Betrachtet man die Verteilung der NASA-TLX-Kennwerte, so lässt sich bei Aufgabe 5 ein klarer Anstieg der Kennwerte geistige Anstrengung, zeitlicher Druck, Aufwand und

Frustration erkennen, was auf eine erhöhte Beanspruchung der Probanden hindeutet. Erklärt werden kann dieser Anstieg damit, dass dies die erste Aufgabe des Interaktionsblocks darstellt und die Probanden mit den neuen Bearbeitungsfunktionen konfrontiert wurden. Die Werte des Interaktionsblocks lagen im Schnitt höher als die des Navigations- und Orientierungsblocks. Probleme mit der Bearbeitung der Elemente gab es keine – Aufgabe 5 wurde von allen Probanden korrekt gelöst, ebenso gab es keine Kritik in den Diskussionen. Das Konzept der Detailansicht konnte somit bekräftigt werden.

Die Auswertung der Statistik zusammen mit den Aussagen der Probanden ergab jedoch, dass die Knotenauswahl im Prototyp bei der Bedienung des Smartphones ungewohnt sei. Bei der Auswahl der Knoten über das bereit gestellte Auswahlrechteck kam es bei der Mehrheit der Probanden zu Problemen. Ein Proband meinte, der „Sinn des Markierungsrahmens sei nicht von Beginn an klar“, ein anderer meinte „er habe bisher in noch keiner (Handy-)App ein ähnliches Auswahl-Rechteck gesehen“. Auf Nachfrage gaben 9 der 11 Probanden an, dass sie Probleme mit dieser Art der Auswahl hatten und eine andere Methode zur Knotenauswahl bevorzugen würden. Die Probleme gründen sich auf den folgenden Aspekten:

- Das Auswahlrechteck liegt nicht zentral unter dem Berührungspunkt sondern leicht links davon, damit der Finger den Inhalt nicht verdeckt. Dies sorgte jedoch für Verwirrung, da die Probanden gewohnt waren, die Objekte direkt per Berührung auszuwählen oder zu manipulieren. Dies war besonders ausgeprägt bei Probanden, die ein Smartphone besaßen und bereits dessen Bedienung gewohnt waren. Die Gewohnheitsmuster werden in Kapitel 7.5.3 genauer diskutiert.
- Aus selbigem Grund hatten Linkshänder erhebliche Probleme mit der Bedienung des Auswahlrechtecks, da der Inhalt beim Berühren unter ihrem Finger verschwand.
- Durch den Versatz kam es zu Problemen bei der Auswahl von Knoten, die sich am rechten Bildschirmrand befanden, da man mit dem Finger bereits über den Bildschirmrand hinaus kam, um das Rechteck korrekt zu platzieren.
- Auf Grund der prototypischen Implementierung kam es vor, dass das Rechteck mit einer leichten Verzögerung an die gewünschte Position trat, was die Bedienbarkeit negativ beeinflusste.

Auf Nachfrage brachten die Probanden folgende Verbesserungsvorschläge ein:

1. Auswahlrechteck zentral unter dem Finger platzieren. Den Wechsel in die Detailansicht über das Tab beibehalten.
2. Direkte Auswahl des Knotens per Berührung. Dieses Konzept wurde von der Mehrheit der Probanden als Lösungsvorschlag genannt, da sie es von anderen Smartphone-Anwendungen bereits gewohnt sind. Bei diesem Konzept kann optional bei der Auswahl ein automatischer Wechsel zur Detailansicht erfolgen.

Die Bediengewohnheiten der Probanden spielen somit eine große Rolle, da diese bereits stark durch andere Smartphone-Anwendungen geprägt sind, was in Kapitel 7.5.3 nochmals

vertieft wird. Die Mehrheit der Probanden wünscht sich die Umsetzung eines Konzepts mit direkter Manipulation, das in 7.6 festgehalten wird.

Das Einfügen von Knoten aus Aufgabe 6 stellte für die Probanden keine Probleme dar, was man anhand der niedrigen Werte der NASA-TLX-Diagramme erkennen kann, sowie der korrekten Bearbeitung. Die erste Teilaufgabe benötigte mehr Zeit, da die Probanden sich an die Methode gewöhnen mussten und mehrere Knoten einfügen sollten. Lediglich ein Proband merkte an, dass er nach der Einfüge-Interaktion gerne den Knoten in der Graphenübersicht sehen würde. Dies wurde in Kapitel 7.6 mit aufgenommen.

Zu Aufgabe 7, dem Löschen von Elementen, gab es wenig Feedback durch die Probanden. Es gab offenbar keine Schwierigkeiten bei der Durchführung, die Korrektheit lag im Schnitt bei 90 Prozent. Ein Proband merkte an, dass die Buttons zum Entfernen etwas klein sind und somit schwer zu treffen. Dies kann in Kapitel 7.6 berücksichtigt werden. Die Auswertung des NASA-TLX ergab ebenfalls keine Auffälligkeiten - die Werte lagen allesamt im Mittelfeld der Interaktionsaufgaben. Die eigene Leistung wurde mit 3,2 als gut eingeschätzt. Das Konzept zum Löschen von Elementen wird somit beibehalten.

Aufgabe 8 wies vergleichsweise niedrige und gute Werte auf bei der Auswertung des NASA-TLX. Die Aufgaben wurden korrekt gelöst, die Probanden benötigten im Mittel 35 Sekunden Zeit. Das Hinzufügen von Emittern konnte problemlos umgesetzt werden, die Interaktion wird somit beibehalten.

Betrachtet man die Auswertung des NASA-TLX Fragebogens, so fällt auf, dass Aufgabe 9 die anspruchsvollste Aufgabe der Studie darstellt. Im Mittel bewerteten die Probanden die geistige Anstrengung und den benötigten Aufwand mit 9,2 respektive 9,3. Gleichzeitig lag der Grad der Frustration mit 6,5 vergleichsweise hoch. Aus den Gesprächen nach der Durchführung der Studie ging hervor, dass die Probanden den Ansichtswechsel während der Durchführung der Verbund-Aktion als störend empfanden. 60 Prozent der Probanden gaben dies als Problem an. Die erste Teilaufgabe wurde nur von 70 Prozent der Probanden korrekt gelöst. Aufgabe war es, einen Knoten nach oben hin über den Rezeptor zum Emitter des Zielknotens zu verbinden. Dies bereitete manchen Probanden Probleme, sie führten das Verbinden falsch herum aus und haben Rezeptor und Emitter der Knoten vertauscht. Ein Grund hierfür könnte sein, dass die Probanden das Verbinden intuitiv vom Vaterknoten zum Kindknoten durchführen, und nicht umgekehrt. Dies sollte in den Verbesserungen berücksichtigt werden.

Auf Nachfrage, wie die betroffenen Probanden sich das Verbinden vorstellen, wurden verschiedene Verbesserungsvorschläge unterbreitet. Ein Proband schlug vor, das Verbinden von Knoten direkt in der Minimap des Graphen mittels Ziehen und Ablegen zu realisieren. Hierbei muss jedoch eine Möglichkeit zur Auswahl des gewünschten Konnektors vorgesehen werden, da die Konnektoren selbst zu klein sind, um sie per Berührung gut bedienen zu können. Eine Möglichkeit wäre die Auswahl über eine interaktive Liste, die nach Ziehen der Kante entsteht, oder über vergrößerte Interaktionsbereiche von Emittern. Man könnte sich bei der Applikation 4.4.1 bedienen, die eine Kante vom derzeit ausgewählten Knoten zieht. Auch muss die Kante während des Verbindens für den Benutzer als Vorschau

sichtbar sein, damit dieser eine Rückmeldung über seine Interaktion erhält. Ist nur ein Konnektor vorhanden, wie es bei Rezeptoren häufig der Fall ist, könnte diese Auswahl ausbleiben und die Kante direkt angeschlossen werden.

Ein anderer Proband meinte, man könne den Interaktionsstart einer Verknüpfung in der Detailansicht belassen und lediglich das Anschließen der Kante selbst in der Minimap realisieren. So könnte man explizit den Startkonnektor und mit einer interaktiven Liste den Zielkonnektor wählen. Zudem würde man sich einen Sichtwechsel sparen. Beide Konzepte schließen somit aus, das Anschließen von gezogenen Kanten in der Detailansicht zu realisieren. Bei dieser Variante sollte jedoch das Verbindungssymbol zur Auswahl eines Startkonnektors nicht auf dem Konnektor selbst dargestellt werden, da dies die Auswahl der Konnektoren behindert und bei der Studie mehrmals zur unbeabsichtigten Auswahl eines Konnektors führte. Der Proband schlug vor, den Knopf in den Inhaltsbereich des Emitters hineinzuziehen, oder den Aktionsstart über einen langen Druck auf den Konnektor auszulösen.

Der Sichtwechsel bei Durchführung der Knoten-Verknüpfung ist für die Mehrheit der Probanden nach eigener Angabe zu verwirrend und zu umständlich, was sich auch sehr gut an der Bewertung des NASA-TLX-Fragebogens heraus gestellt hat. Die Implementierung dieser Aktion wird somit wie in Kapitel 7.6 beschrieben abgeändert, um den Probanden ein besseres Bedienerlebnis zu gewährleisten.

7.5.3 Bedienung

Es hat sich heraus gestellt, dass die Bediengewohnheiten der Probanden großen Einfluss auf die Umsetzbarkeit der gestellten Aufgaben hatte. Benutzer eines Smartphones sind beispielsweise bereits mit der Stapel-orientierten Aufrufart des Systems vertraut, wie sie in Kapitel 2.3.2 beschrieben ist. Die Probanden haben erwartet, dass sie mit der „Zurück“-Taste jederzeit in die vorige Ansicht springen können, da es dem gängigen Navigations-Konzept anderer Smartphone-Anwendungen entspräche. Dies war jedoch durch das Tab-Prinzip nicht vorgesehen. So haben fünfzig Prozent der Probanden versehentlich in der Detailansicht die „Zurück“-Taste betätigt, anstatt über die Tabs zu navigieren und das Programm somit unbeabsichtigt beendet. Dies sorgte bei fast allen Probanden für Frustration.

Gleichzeitig merkte ein Proband an, dass „die Verwendung der Tabs nicht der gewohnten Verwendung entspricht, was Anfangs ungewohnt sei“. Er habe erwartet, dass ein Tab, insbesondere die Detailansicht, immer den selben Inhalt anzeigt. Die betroffenen Probanden haben die Umsetzung des Stack-Prinzips nach der Studie als Verbesserungsvorschlag angebracht. Das vorgeschlagene Konzept sieht vor, durch einfaches oder langes Drücken eines Knotens in dessen Detailansicht zu wechseln und mit der „Zurück“-Taste wieder zurück in die Minimap. Dies kann ebenso auf den Ladebildschirm übertragen werden, womit die Tab-Leiste zu Gunsten von mehr freier Bildschirmfläche entfernt werden könnte. Es ist jedoch ein Konzept zu finden, mit dem man den Graphen speichern kann, bevor dieser vom Activity-Stack entfernt wird, beispielsweise durch eine Benutzerabfrage.

Das vorgeschlagene Konzept würde ebenso dem Navigations-Aspekt aus Kapitel 7.5.1 entgegenkommen, da es eine direkte Knotenauswahl implementiert.

Eine weitere gewohnte Funktion ist die Navigation durch eine Zoom-Funktion, welche gängiger-weise über eine Multitouch-Geste mit zwei Fingern ausgeführt wird. Hierbei kann ein Text oder eine Grafik durch auseinanderziehen der Berührungspunkte vergrößert und durch zusammenführen wieder verkleinert werden [2]. Das Konzept findet in vielen mobilen Browsern Anwendung. Das Zoom-Konzept wurde bereits in Kapitel 7.5.1 diskutiert.

7.6 Verbesserungen

Im Folgenden werden alle Verbesserungen aufgeführt, die sich aus der Auswertung ergeben haben. Als Leitprinzip werden die Bediengewohnheiten der Probanden, sowie gängige Bedienmuster des verwendeten Smartphone-Systems herangezogen. Der Benutzer soll im Vordergrund stehen und sich nicht an die Bedienung der Applikation anpassen müssen. Ungewohnte Bedienmuster, die in der Studie zu Problemen geführt haben, sollen durch gängige ersetzt werden, um dem Benutzer bei der Bedienung ein vertrautes Gefühl zu geben. Kleinere Änderungen werden ebenfalls aufgeführt. Die Umsetzung erfolgt soweit diese noch im Rahmen der Diplomarbeit möglich sind. Umfangreiche Änderungen sind Teil von Kapitel 8.1.

7.6.1 Verbesserter Arbeitsfluss

Um den Arbeitsfluss zu verbessere, sind folgende Änderungen denkbar:

- Nach dem Hinzufügen eines Emitters soll dieser für einen besseren Arbeitsfluss gleich zum Editieren ausgewählt werden
- Nach dem Hinzufügen eines Knotens kann optional die Graphübersicht angezeigt werden, in der der Knoten markiert wird. Somit sieht der Benutzer direkt, dass ein Knoten hinzugefügt wurde und an welcher Position.
- Das Verbinden von Knoten durch eine Kante kann wie in 7.5.2 beschrieben verbessert werden, da das jetzige Konzept zu umständlich ist.

7.6.2 Verbesserte Darstellung und Bedienung

Zur Verbesserung der Darstellung können die folgenden Aspekte implementiert werden;

- Die Darstellung breiter Graphen könnte durch die Implementierung einer Landscape-Ansicht verbessert werden. Hierzu muss die Orientierung des Smartphones berücksichtigt werden, durch deren Wechsel eine Neuberechnung der Ansicht erfolgt.
- Die Größe der Buttons zum Entfernen von Elementen kann vergrößert werden, damit die Nutzer diese besser betätigen können

7.6.3 Verbesserte Gestensteuerung

Da einige Probanden Probleme mit der Auswahl kleiner Elemente hatten, kann die Gestensteuerung der Anwendung verbessert werden, um diese Interaktionsart besser zu unterstützen. Im vorgestellten Konzept werden Gesten lediglich für den Wechsel des derzeit betrachteten Knotens in der Detailansicht verwendet, siehe Kapitel 5.1.2. Dieses wird jedoch nicht durch visuelles Feedback unterstützt, weshalb in der Studie auch kein Proband auf die Funktion gestoßen ist.

Das vorgestellte Konzept kann wie folgt modifiziert werden, um dem Benutzer mehr Feedback zu liefern und gleichzeitig die Auswahl kleiner Elemente, wie der Flows, zu erleichtern. Die horizontale Navigation durch Wischen soll zur Auswahl eines einzigen Flows benutzt werden, statt den Knoten direkt zu wechseln. Hierbei soll im Grundzustand immer genau ein Flow ausgewählt sein, um dem Benutzer visuelles Feedback zu liefern, welches Element er markiert hat. Durch eine horizontale Wisch-Bewegung nach links oder rechts soll der jeweils nächste Flow in der Liste der Emitter, respektive Rezeptoren, markiert werden. Durch eine vertikale Wisch-Bewegung soll der ausgewählte Flow von der Rezeptor-Ansicht zur Emitter-Ansicht wechseln. Bei der Auswahl soll zusätzlich der Emitter, an den der ausgewählte Flow angeschlossen ist, ausgewählt werden. Somit haben die Benutzer gleichzeitig die Möglichkeit, über die Gestensteuerung einen Emitter auszuwählen.

Möchte der Benutzer nun den Knoten wechseln, kann er nach wie vor über eine vertikale Wisch-Bewegung den vorigen oder nachfolgenden Knoten erreichen. Hierbei wird nun die Kante verfolgt, die zuvor über die Wisch-Navigation ausgewählt wurde. Zudem muss die gewünschte Kante sich in der Richtung befinden, in welcher der Nutzer navigieren möchte. Um möglichst viele Nutzer mit dem Bedienkonzept ansprechen zu können, soll man die Kante jedoch nach wie vor über eine Berührung auswählen können.

Nach links wischen Auswahl des Flows links vom derzeit ausgewählten Flow

Nach rechts wischen Auswahl des Flows rechts vom derzeit ausgewählten Flow

Nach oben wischen, Emitter-Flow ausgewählt Auswahl des ersten Flows in der Rezeptor-Liste

Nach oben wischen, Rezeptor-Flow bereits ausgewählt Folgen des ausgewählten eingehenden Flows

Nach unten wischen, Rezeptor-Flow ausgewählt Auswahl des ersten Flows in der Emitter-Liste

Nach unten wischen, Emitter-Flow bereits ausgewählt Folgen des ausgewählten ausgehenden Flows

Um dem Benutzer anzuzeigen, dass er die Gestensteuerung zur Auswahl und Navigation nutzen kann, sollte noch ein zusätzliches Hinweiskonzept eingeführt werden. Man könnte beispielsweise bei Berührung des Bildschirms transparente Pfeile einblenden, die die

möglichen Interaktions-Richtungen beschreiben. Bei Ausführung kann die durchgeführte Wisch-Aktion mit dem Hervorheben des Richtungspfeils unterstützt werden.

7.7 Fazit

Die Evaluierung des entwickelten Konzeptes brachte folgende Erkenntnisse:

Die Durchführbarkeit der Aufgaben war gegeben, es gab keine unlösbaren Probleme während der Benutzerstudie. Somit wurde der Funktionsumfang von Navigation & Bearbeitung abgedeckt. Hypothese eins aus 5.2 wurde bestätigt.

Die Übersicht im Graphen ist gegeben, die Darstellung der Knoten in der Übersicht ist klar und wird beibehalten. Dadurch war eine rasche Navigation gegeben. Hypothese zwei aus Kapitel 5.2 wurde bestätigt.

Die Orientierung innerhalb und zwischen den Graphendarstellungen funktioniert reibungslos. Das Konzept ist geeignet, um die kooperative Arbeit mit der Host-Applikation aufzunehmen. Hypothese drei aus Kapitel 5.2 wurde bestätigt.

Die Detailansicht wird beibehalten, da sich die Bearbeitung der Knotenelemente als intuitiv und funktional herausgestellt hat. Hypothese vier aus Kapitel 5.2 wurde bestätigt.

Der Workflow bei der Ausführung von Interaktionen muss gegeben sein. Dies ist bei der Mehrheit der Interaktionen der Fall. Das Verbinden von Kanten muss jedoch vereinfacht werden. Einige der Interaktionskonzepte können durch automatische Status-Übergänge vereinfacht werden – ein Sichtwechsel, während der Ausführung von Interaktionen, soll vermieden werden. Somit muss für Hypothese fünf aus Kapitel 5.2 an diesen Stellen nachgebessert werden, die Hypothese wurde nicht vollständig erfüllt.

Die Knotenauswahl sollte überarbeitet werden. Sie kann durch ein Touch-Konzept ersetzt werden, da die Mehrheit der Probanden sich dies gewünscht haben und das Konzept im Prototyp erwartet wurde.

Eine Zoomfunktion kann für eine gute Lesbarkeit in die Graphenübersicht eingebaut werden, um auch Benutzern mit einer Sehschwäche das Lesen der Knotenattribute zu ermöglichen. Die Zoomfunktion soll hierbei jedoch keine Voraussetzung für die Bedienung des Graphen darstellen, Knoten sollen aus jeder beliebigen Distanz auswählbar sein. Diese kann auch auf festen Zoomstufen beruhen, beispielsweise ein Hineinzoomen auf eine gut lesbare Textgröße als eine Art Lupenfunktion.

Bediengewohnheiten der Nutzer müssen berücksichtigt werden; gängige Smartphone-Konzepte prägen die Erwartungshaltung bei der Erstbedienung. Ein Stack-Prinzip soll die Verwendung der Tab-Leiste ersetzen, um Benutzern die Navigation mit dem „Zurück“-Knopf zu ermöglichen.

8 Zusammenfassung

In der Diplomarbeit „Visualisierung von Filter/Flow-Graphen auf mobilen Endgeräten“ wurde ein Konzept zur ansprechenden Darstellung und Bearbeitung auf Smartphones mit dem Betriebssystem Android entwickelt. Das Konzept wurde anhand eines Prototyps belegt. Eine Benutzerstudie hat gezeigt, dass das Konzept geeignet ist, um Filter/Flow-Graphen übersichtlich darzustellen, sie einfach mit Hilfe des Touchscreens zu editieren und mit einem großen Abbild des Graphen parallel zu arbeiten. Durch die Verwendung von *Mono for Android* ist zudem Portabilität gegeben, wodurch das Konzept und Teile des Prototyps auch auf anderen Mobilplattformen umsetzbar ist. Aus der Benutzerstudie ging hervor, dass die Bediengewohnheiten der Probanden eine große Rolle spielen. Diese sollten deshalb bei einer möglichen Portierung unbedingt beachtet und für die Zielplattform angepasst werden.

8.1 Ausblick

Es wurde aufgezeigt, dass das entwickelte Konzept für derzeitige Smartphone-Systeme geeignet ist, um Graphen darzustellen und bearbeiten zu können. Der Trend geht jedoch nicht nur in Richtung Smartphones, sondern auch in Richtung Tablets - Geräte wie das *iPad* ermöglichen durch ihr größeres Display und ihre höhere Auflösung die Darstellung größerer Graphen und von mehr Details. Da es auch Tablets mit dem Betriebssystem Android gibt, wären Konzepte zu prüfen, die das Display für eine Darstellung von Graph und Minimap in Einem ermöglichen, beispielsweise als Overlay. Des Weiteren könnte die Software auf weitere Mobilsysteme portiert werden, zum Beispiel *Windows Phone 8*, bei dem der Quell-Code größtenteils übernommen werden kann.

Um das Konzept weiterzuentwickeln, können die vorgeschlagenen Verbesserungen aus Kapitel 7.6 implementiert werden, die die Gewohnheiten der getesteten Smartphone-Benutzer miteinbeziehen. Anschließend kann eine quantitative Studie erfolgen, bei der das verbesserte Konzept gegen ein Zoom-Mockup antreten kann. Hierbei wären Grenzfälle mit großen Graphen zu betrachten.

Des Weiteren kann die kooperative Arbeit zwischen der Mobilanwendung und der Host-Applikation untersucht werden. Die Bearbeitung eines Graphen könnte hierbei gemeinsam an einer großen Darstellung oder verteilt auf verschiedene Endgeräte erfolgen. Es kann beispielsweise untersucht werden, welche Funktionen für eine reibungslose Kommunikation zwischen den Teilnehmern notwendig sind. Hierbei ist auch eine Anwendung für große berührungsempfindliche Displays wie das Microsoft PixelSense (ehemals Microsoft Surface)

denkbar, bei der Konzepte für die omnidirektionale Bearbeitung von Graphen zum Tragen kommen. Die vorliegende Mobilanwendung könnte auch als nutzerabhängige Erweiterung der Host-Anwendung verwendet werden. Ähnliche Konzepte zur Interaktion bei der kooperativen Arbeit an Visualisierungen wurden bereits bei der Kommunikation zwischen dem Microsoft PixelSense und einer Powerwall von Ploner [23] und Püttmann [25] erarbeitet.

9 Anhang

Gerätemerkmal	Ausstattung
Bildschirmtyp	Super AMOLED HD-Display
Bildschirmgröße	4,65-Zoll (11,8 cm)
Auflösung	1280 x 720 Pixel
Eingabe	Kapazitiver Touchscreen, 3 kapazitive Tasten
Betriebssystem	Android 4.1.1 Jelly Bean
Mobilfunknetze	Triband-UMTS (900, 1700, 2100 MHz), HSDPA, HSUPA, Quadband-GSM
Übertragungsstandards	WLAN IEEE 802.11 b/g/n, Bluetooth 2.1 + EDR, Near Field Communication
Interner Speicher	16 GB Flash-Speicher + 1 GB RAM
Prozessor	1,2-GHz Dual-Core-Prozessor
Schnittstellen	Micro-USB 2.0, 3,5-mm-Klinke
Kamera	Rückseitig 5,0 Megapixel (2560 x 1920 Pixel) mit Autofokus, 1,3 Megapixel (1.280 x 1.024 Pixel) auf der Vorderseite
Weitere Sensoren	A-GPS-Empfänger, digitaler Kompass, Beschleunigungssensor, Näherungssensor, Umgebungslichtsensor, 3-Achsen-Gyroskop, Barometer
Akku-Kapazität	1750 mAh

Tabelle 9.1: Spezifikationen des Galaxy Nexus. Quelle: [6]

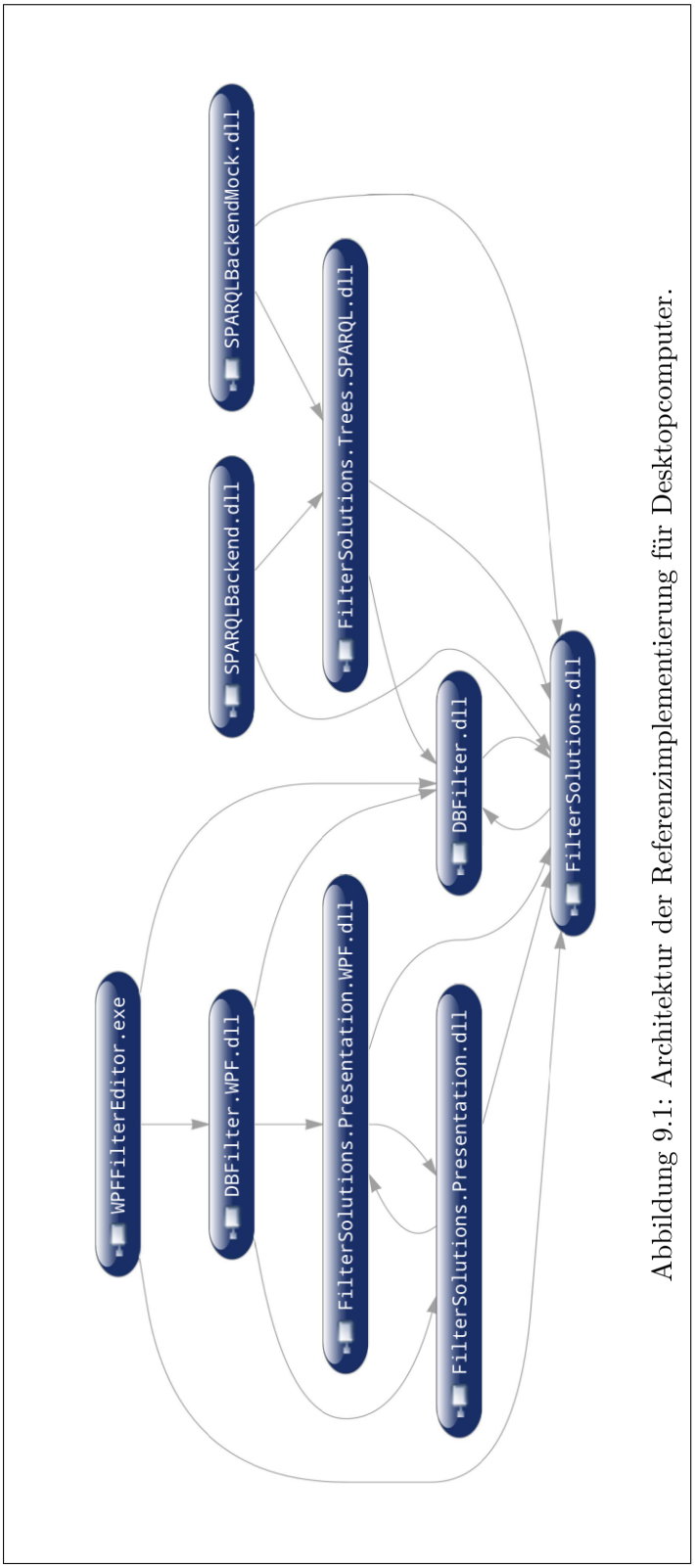


Abbildung 9.1: Architektur der Referenzimplementierung für Desktopcomputer.

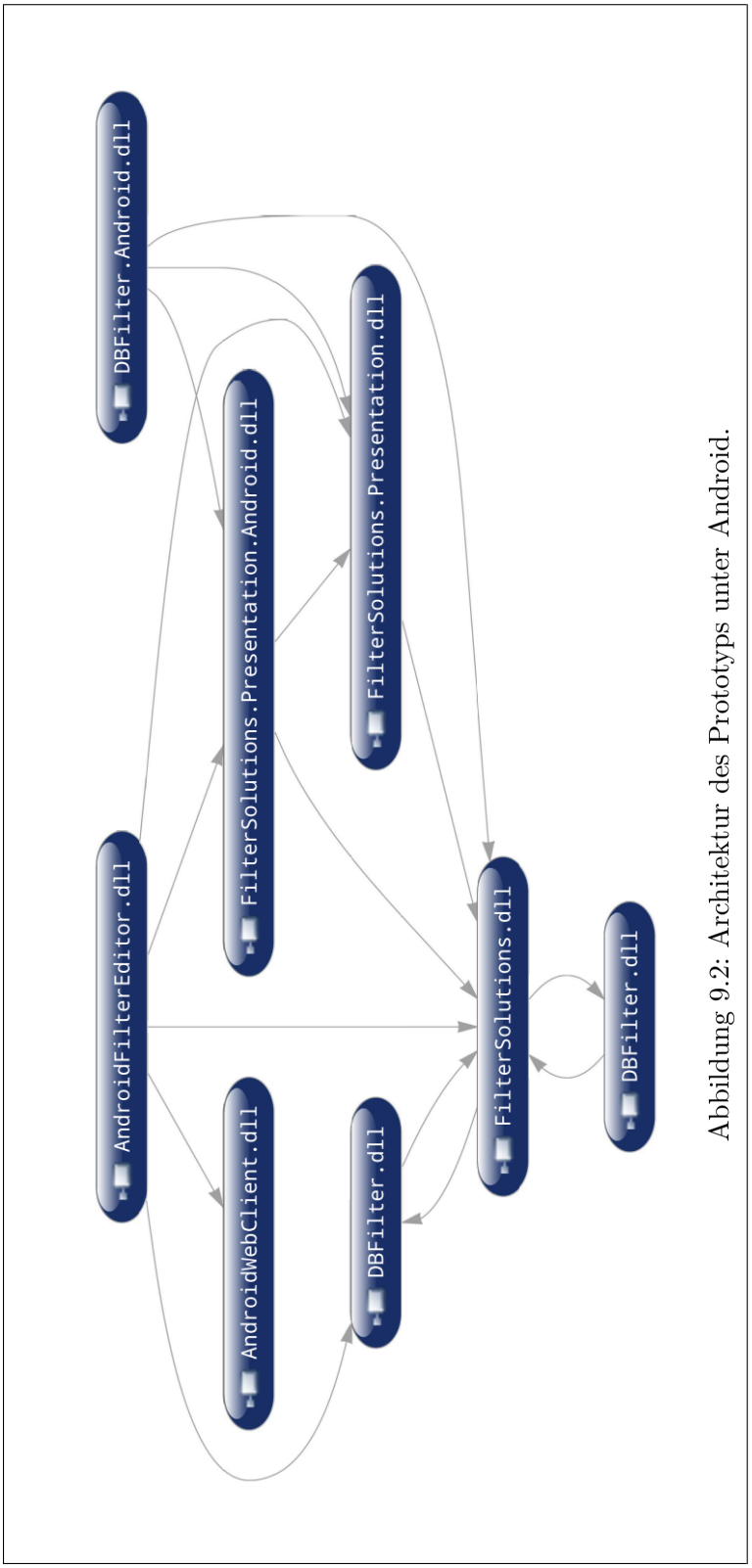
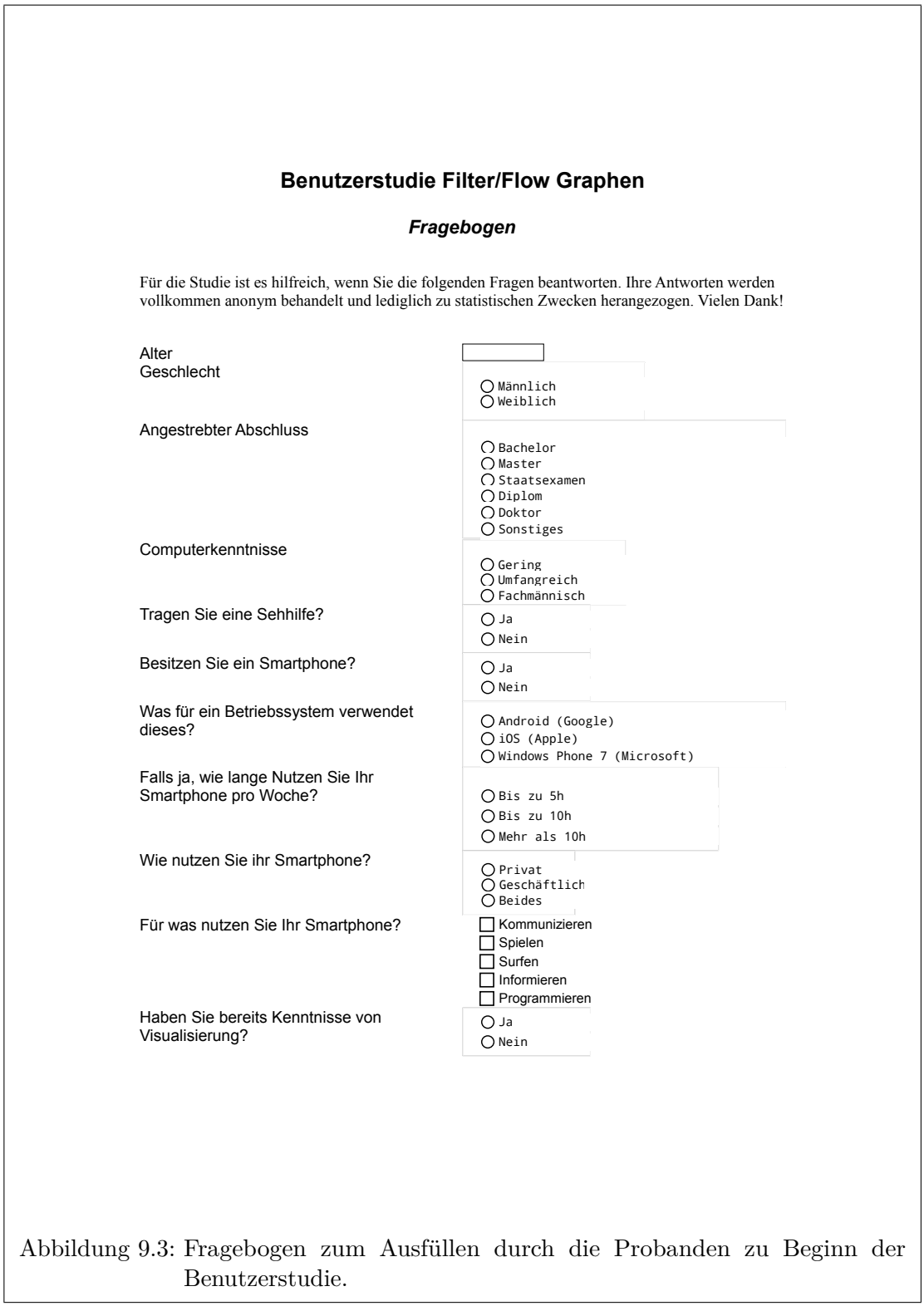


Abbildung 9.2: Architektur des Prototyps unter Android.



Benutzerstudie Filter/Flow Graphen

Fragebogen

Für die Studie ist es hilfreich, wenn Sie die folgenden Fragen beantworten. Ihre Antworten werden vollkommen anonym behandelt und lediglich zu statistischen Zwecken herangezogen. Vielen Dank!

Alter

Geschlecht
- ○ Männlich
- ○ Weiblich

Angestrebter Abschluss
- ○ Bachelor
- ○ Master
- ○ Staatsexamen
- ○ Diplom
- ○ Doktor
- ○ Sonstiges

Computerkenntnisse
- ○ Gering
- ○ Umfangreich
- ○ Fachmännisch

Tragen Sie eine Sehhilfe?
- ○ Ja
- ○ Nein

Besitzen Sie ein Smartphone?
- ○ Ja
- ○ Nein

Was für ein Betriebssystem verwendet dieses?
- ○ Android (Google)
- ○ iOS (Apple)
- ○ Windows Phone 7 (Microsoft)

Falls ja, wie lange Nutzen Sie Ihr Smartphone pro Woche?
- ○ Bis zu 5h
- ○ Bis zu 10h
- ○ Mehr als 10h

Wie nutzen Sie ihr Smartphone?
- ○ Privat
- ○ Geschäftlich
- ○ Beides

Für was nutzen Sie Ihr Smartphone?
- ☐ Kommunizieren
- ☐ Spielen
- ☐ Surfen
- ☐ Informieren
- ☐ Programmieren

Haben Sie bereits Kenntnisse von Visualisierung?
- ○ Ja
- ○ Nein

Abbildung 9.3: Fragebogen zum Ausfüllen durch die Probanden zu Beginn der Benutzerstudie.

Auswertung Bearbeitungsdauer

Proband-Nr.	1a	1b	1c	2a	2b	2c	3a	3b	3c	4a	4b	4c	5a	5b	6a	6b
1	17,698	8,527	28,411	37,625	60,054	33,695	14,440	43,679	14,263	16,168	12,262	10,611	46,402	51,774	80,999	80,404
2	15,544	10,038	10,936	17,292	37,434	33,399	19,385	40,575	34,239	15,329	12,044	10,225	51,578	49,238	71,447	19,532
3	15,736	9,743	28,437	13,467	20,277	13,147	27,527	17,966	20,550	15,094	17,620	16,230	86,122	50,699	55,343	47,320
4	22,575	15,538	8,718	10,172	21,474	17,424	18,165	14,297	21,693	21,635	27,471	15,984	40,839	46,575	56,807	23,433
5	20,204	14,309	9,545	16,787	17,950	26,630	9,804	28,351	39,976	10,551	9,451	15,419	36,278	42,887	98,207	40,504
6	26,716	9,986	11,466	21,917	27,319	50,009	18,551	20,938	31,590	8,939	8,234	9,063	52,413	47,934	112,269	59,750
7	14,621	15,946	7,635	25,027	16,437	44,833	12,659	13,805	14,077	15,357	11,806	18,234	39,148	39,501	61,437	25,051
8	11,272	17,099	64,983	20,216	22,612	26,419	21,686	30,135	12,996	12,708	11,768	10,975	59,524	49,837	74,954	23,456
9	27,508	8,580	49,608	14,752	15,256	42,002	10,942	21,999	27,716	8,153	13,702	5,919	49,743	67,284	77,295	27,864
10	30,589	15,349	46,565	15,017	17,498	24,737	51,159	19,793	21,810	14,673	20,342	13,709	34,337	44,959	58,440	21,266
11	23,501	19,762	27,653	13,457	13,771	34,449	18,294	16,440	22,135	18,026	19,295	8,468	49,626	34,921	83,453	24,010
Gemittelte Zeit	**20,54**	**13,17**	**26,72**	**18,7**	**24,55**	**31,52**	**20,24**	**24,36**	**23,73**	**14,24**	**14,91**	**12,26**	**49,64**	**47,78**	**75,51**	**35,69**
Median	**20,2**	**14,31**	**27,65**	**16,79**	**20,28**	**33,4**	**18,29**	**20,94**	**21,81**	**15,09**	**12,26**	**10,97**	**49,63**	**47,93**	**74,95**	**25,05**

Proband-Nr.	7a	7b	8a	8b	9a	9b
1	35,377	76,591	44,448	27,525	###	71,256
2	31,072	39,692	33,317	16,182	97,872	21,787
3	13,559	77,056	37,204	20,915	95,064	36,304
4	38,952	34,400	36,724	17,458	47,150	27,236
5	21,550	46,701	53,496	27,859	94,784	53,925
6	68,979	57,744	74,361	39,879	77,898	76,404
7	25,007	32,222	50,886	23,174	42,972	57,427
8	61,774	53,380	44,480	24,811	64,937	30,684
9	29,409	48,517	48,180	58,440	###	62,440
10	47,860	41,065	36,011	20,606	65,924	26,532
11	35,296	68,958	49,113	22,330	80,941	40,201
Gemittelte Zeit	**37,17**	**52,39**	**46,2**	**27,2**	**82,84**	**45,84**
Median	**35,3**	**48,52**	**44,48**	**23,17**	**80,94**	**40,2**

Page 1

Abbildung 9.4: Bearbeitungsdauer aller Aufgaben aus der Benutzerstudie.

Literaturverzeichnis

[1] 37SIGNALS: *Helpful distortion at NYC & London subway maps.* http://37signals.com/svn/posts/396-helpful-distortion-at-nyc-london-subway-maps. Version: April 2007

[2] BLASCHECK, Tanja ; BOLD, David ; MUHLER, Dominik: *Fachstudie Interaktionskonzepte für Multi-Touch.* Universität Stuttgart, Februar 2011

[3] BOYLE, J. ; BURY, K. ; EVEY, R.: Two studies evaluating learning and use of QBE and SQL. In: *Proceedings of the Human Factors Society 27th Annual Meeting* (1983), S. 663–667

[4] BRANDES, Ulrik: *Layout of Graph Visualizations.* Universität Konstanz, Juni 1999

[5] BRAUN, Paul: Smartphone Owners Want Thin Devices with Larger Displays. In: *Strategy Analytics* (2012), März, 14. http://www.strategyanalytics.com/default.aspx?mod=reportabstractviewer&a0=7194

[6] DIVERSE: *Galaxy Nexus - Technische Daten.* www.google.de. http://www.google.de/nexus/tech-specs.html. Version: Mai 2012

[7] FERBER, Thomas ; JANSA, Paul ; DILLI, Johannes: *Fachstudie Security & Privacy – Wie machen das Apple, Google und Co.? : eine Sicherheitsbetrachtung mobiler Endgeräte.* Universität Stuttgart, Februar 2012

[8] FERSTL, Otto K. ; SINZ, Elmar J.: *Grundlagen der Wirtschaftsinformatik, Band 1.* Bd. 1. 5. Auflage. Oldenbourg Verlag, 2006. – 495 S.

[9] GOOGLE: *Activities.* Android SDK. http://developer.android.com/guide/components/activities.html. Version: September 2012

[10] GOOGLE: *Canvas and Drawables.* Android SDK. http://developer.android.com/guide/topics/graphics/2d-graphics.html. Version: Mai 2012

[11] GOOGLE: *Layouts.* Android SDK. http://developer.android.com/guide/topics/ui/declaring-layout.html. Version: Oktober 2012

[12] GOOGLE: *Public Class Canvas.* Android SDK. http://developer.android.com/reference/android/graphics/Canvas.html. Version: Mai 2012

[13] GREENE, S.L. ; DEVLIN, S.J. ; CANNATA, P.E. ; L.M.GOMEZ: No IFs, ANDs, or ORs: A study of database querying. In: *International Journal of Man-Machine Studies* 32 (1990), März, S. 303–326

[14] Haag, Florian ; Lohmann, Steffen ; Ertl, Thomas: Simplifying Filter/Flow Graphs by Subgraph Substitution. In: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2012)* (2012), Oktober, S. 145–148

[15] Ham, Frank V. ; Perer, Adam: Search, Show Context, Expand on Demand: Supporting Large Graph Exploration with Degree-of-Interest. In: *IEEE Transactions on visualization and computer graphics* 15 (2009), Dezember, S. 953–960

[16] Hart, Sandra G.: NASA-Task Load Index (NASA-TLX); 20 Years Later. (2006), 5. `http://humansystems.arc.nasa.gov/groups/TLX/downloads/HFES_2006_Paper.pdf`

[17] Hart, Sandra G. ; Staveland, Lowell E.: Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. (1986), 46. `http://humansystems.arc.nasa.gov/groups/TLX/downloads/NASA-TLXChapter.pdf`

[18] Lazar, Jonathan ; Feng, Jinjuan H. ; Hochheiser, Harry: *Research Methods - In Human-Computer Interaction*. Wiley, 2010. – 448 S.

[19] Michard, A.: A new database query language for non-professional users: Design principles and ergonomic evaluation. In: *Behavioral and Information Technology* 13 (1982), S. 279–288

[20] Microsoft: *Microsoft Presentaiton Foundation*. MSDN. `http://msdn.microsoft.com/de-de/library/vstudio/ms754130.aspx`. Version: Oktober 2012

[21] Mindjet: *Mindjet for Android*. `http://www.mindjet.com/products/mindmanager#mmMobiApp`. Version: Oktober 2012

[22] Parhi, Pekka ; Karlson, Amy K. ; Bederson, Benjamin B.: Target Size Study for One-Handed Thumb Use on Small Touchscreen Devices. In: *MobileHCI'06* (2006), September, S. 203–210

[23] Ploner, Nico: *Gestensteuerung für Powerwall-basierte Visualisierungen*, Universität Stuttgart, Diplomarbeit, 2012

[24] Prud'hommeaux, Eric ; Seaborne, Andy: *SPARQL Query Language for RDF*. W3C. `http://www.w3.org/TR/rdf-sparql-query/`. Version: Januar 2008

[25] Püttman, Edwin: *Tabletop-Computer-basierte Steuerung für Powerwall-Visualisierungen*, Universität Stuttgart, Diplomarbeit, 2012

[26] Reisner, Phyllis ; Boyce, Raymond F. ; Chamberlin, Donald D.: Human factors evaluation of two data base query languages: square and sequel. In: *National Computer Conference, 1975* (1975), Mai, S. 447–452

[27] Roto, Virpi ; Popescu, Andrei ; Koivisto, Antti ; Vartiainen, Elina: Minimap – A Web Page Visualization Method for Mobile Phones. In: *CHI '06 Proceedings of the SIGCHI conference on Human Factors in computing systems* (2006), April, S. 35–44

[28] Schumann, Heidrun ; Müller, Wolfgang: *Visualisierung - Grundlagen und allgemeine Methoden.* Springer, 2000. – 388 S.

[29] Shneiderman, Ben: Visual User Interfaces for Information Exploration. In: *Proceeding of the 54th Annual Meeting of The Amercian Society for Information Sciences, vol. 28 (Washington, DC, Oct. 27-31, 1991)* 28 (1991), August, S. 379–384

[30] Shneiderman, Ben: Dynamic Queries for Visual Information Seeking. In: *Software, IEEE* 11 (1994), November, S. 70–77

[31] Shneiderman, Ben: The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In: *IEEE Symposium on Visual Languages, 1996. Proceedings.* (1996), September, S. 336–343

[32] Tamassia, Roberto: Graph Drawing. In: *Lecture Notes in Computer Science* (1997), S. 815–832

[33] Xamarin: *FAQ: Do you have any student or academic discounts?* support.xamarin.com. `http://support.xamarin.com/customer/portal/articles/177042-do-you-have-any-student-or-academic-discounts-`. Version: Juni 2012

[34] Xamarin: *Why you'll love Mono for Android.* xamarin.com. `http://xamarin.com/monoforandroid`. Version: Juni 2012

[35] Young, Degi ; Shneiderman, Ben: A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. In: *Technical Reports of the Computer Science Department, University of Maryland* (1993), Februar, S. 32

Alle URLs wurden zuletzt am 15.10.2012 geprüft.

Glossar

Android SDK Das Android Software Development Kit beinhaltet die APIs jeder verfügbaren Android Version. Des Weiteren bietet es eine Anbindung in die Entwicklungsumgebung Eclipse, einen graphischen Editor für Oberflächen im XML-Format, sowie einen Android Emulator, auf dem entwickelte Applikationen getestet werden können. Android spezifische Treiber, sowie die Android Debugging Bridge zum Zugriff auf das Gerät, werden ebenfalls mitgeliefert. 23, 25

NASA-TLX Der NASA-TLX (NASA Task Load Index) ist ein Hilfsmittel zur Ermittlung der subjektiven Arbeitsbelastung bei der Mensch-Computer Interaktion [17], [16]. Er stellt eine mehrdimensionale Bewertungsprozedur dar, die eine Gesamtbewertung der Arbeitsbelastung durch die gewichtete Summe aus sechs Faktoren berechnet. Der NASA-TLX wird in verschiedenen Anwendungsgebieten erfolgreich angewandt, darunter Überwachungs- und Kontrollumgebungen, Simulationen und Laborversuche. Die Vorlagen zur handschriftlichen Auswertung finden sich auf `http://humansystems.arc.nasa.gov/groups/TLX/paperpencil.html`. 89, 90, 97–99

Operatorpräzedenz Die Operatorpräzedenz, oder auch Operatorrangfolge, beschreibt die Reihenfolge, in der Operatoren in einem Ausdruck ausgewertet werden. Im natürlichen Sprachgebrauch wird dies oft als „Punkt-Vor-Strich Regel“ bezeichnet, die besagt, dass Multiplikation und Division vor Addition und Subtraktion ausgewertet werden müssen. 12, 13

SPARQL SPARQL ist eine Abfragesprache für RDF, einem Datenformat basierend auf gerichteten Graphen um Informationen im Internet zu repräsentieren. Sie wird vom W3C-Konsortium definiert und empfohlen [24]. SPARQL kann dazu benutzt werden, Abfragen auf unterschiedliche Datenquellen zu starten. Die Syntax von SPARQL ähnelt der von *SQL*. Ähnlich wie SQL beinhaltet SPARQL auch die Möglichkeit, Werte zu prüfen und Anfragen mit Restriktionen zu versehen. Die Ergebnismengen einer SPARQL-Anfrage können klassische Mengen sein, oder wiederum RDF-Graphen. Wie andere Abfragesprachen kann auch SPARQL mit Filter-Flow-Graphen visualisiert werden. Die im Prototyp verwendeten Knotentypen aus Kapitel 2.1.5 implementieren einen Teil der Ausdrücke aus SPARQL, um die Funktionsfähigkeit des Konzeptes zu belegen. 78, 79, *siehe* SQL

SQL Die Structured Query Language, kurz *SQL*, ist die dominierende Sprache zur Erstellung und Manipulation relationaler Datenbanken [8]. Sie wird von nahezu

jedem gängigen Datenbankmanagementsystem unterstützt. SQL besteht hierbei aus mehreren Sprachbestandteilen, darunter einer Datenbankanfragesprache, zur Spezifikation von Anfragen an die Datenbasis. Diese kann durch die Anwendung des Filter/Flow-Modells aus 2.1.2 visualisiert werden, um es unerfahrenen Anwendern zu ermöglichen, komplexe Ausdrücke zu erstellen. 123

WPF Windows Presentation Foundation (WPF) bietet Entwicklern ein einheitliches Programmiermodell für die Entwicklung einer umfangreichen Benutzerumgebung für intelligente Windows-Clients, die die Benutzeroberfläche, Medien und Dokumente umfasst [20]. 40, 80–82

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Thomas Ferber)