FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Portierung der prefuse-Funktionalität nach Android

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Dritan Ljatifi
Matrikelnummer 0006706

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Univ.-Prof. Mag. Dr. Silvia Miksch
Mitwirkung: Dipl.-Inf. Dr. Tim Lammarsch

Wien, 16.12.2015

Dritan Ljatifi	Silvia Miksch

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Porting Functionality of prefuse to Android

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering & Internet Computing

by

Dritan Ljatifi
Registration Number 0006706

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Mag. Dr. Silvia Miksch
Assistance : Dipl.-Inf. Dr. Tim Lammarsch

Vienna, 16.12.2015 _____ _____

Dritan Ljatifi Silvia Miksch

Danksagungen

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Diplomarbeit unterstützt und motiviert haben.

Im allererster Stelle bin ich Allah dankbar, Der mir die Kraft, die Gesundheit, die Motivation und alles Nötige gegeben hat, diese Diplomarbeit zur Ende zu bringen.

Ganz besonders gilt mein Dank Frau Prof. Silvia Miksch und Dr. Tim Lammarsch, die meine Arbeit und somit auch mich betreut hat. Nicht nur gaben sie mir immer wieder durch kritisches Hinterfragen wertvolle Hinweise – auch für ihre moralische Unterstützung und kontinuierliche Motivation haben einen großen Teil zur Vollendung dieser Arbeit beigetragen. Sie haben mich dazu gebracht, über meine Grenzen hinaus zu denken. Vielen Dank für die Geduld und Mühen.

Daneben gilt mein Dank meinem Kollegen David Seebacher, mit dem ich Brainstorming geführt habe, wenn ich Ideen für meine Diplomarbeit gebraucht habe.

Zusätzlich bedanke ich mich bei Frau Dipl. Ing. Kerstin Blumenstein, mit der ich das Expert/inneninterview durchgeführt habe.

Auch mein Vorgesetzter und Arbeitskollegen haben maßgeblich daran mitgewirkt, dass diese Diplomarbeit nun in dieser Form vorliegt. Vielen Dank, dass Sie mir den notwendigen Freiraum, den ich für diese Diplomarbeit gebraucht habe, gegeben haben.

Nicht zuletzt gebührt meinen Eltern und meiner ganzen Familie Dank, ohne welche dieses ganze Unternehmen schon im Vorhinein niemals zustande gekommen wäre.

Erklärung zur Verfassung der Arbeit

Dritan Ljatifi Bsc

Johannitergasse 3/24, 1100 Wien

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 16.12.2015

_____

Dritan Ljatifi

Kurzfassung

Informationsvisualisierung hat eine große Bedeutung, um Benutzer/innen die Darstellung komplexer, großer Datenmengen zu ermöglichen. Sie besitzt eine lange Geschichte auf dem Desktop-Computer, und deswegen gibt es bereits zahlreiche Tools und Frameworks, die den Bedarf eines Großteils der Nutzer/innen abdecken. Durch die zunehmende Verbreitung von mobilen Geräten in den letzten Jahren und durch die Verbesserung bei Geräten in Bezug auf Performance und Bildschirmqualität ist der Bedarf an Informationsvisualisierung auf mobilen Geräten gestiegen. Da es sich hierbei um ein neues Gebiet im mobilen Bereich handelt, gibt es relativ wenige Tools und Frameworks dafür. In dieser Arbeit werden zunächst die vorhandenen Tools und Frameworks in Android untersucht. Dann werden deren Funktionalität und Möglichkeiten kurz erläutert.

Im Anschluss wird die Struktur von prefuse, einer der erfolgreichsten Desktop-Frameworks im Informationsvisualisierungsbereich, analysiert. Dabei werden die Informationen, die aus dem State of the Art über prefuse gewonnen wurden, durch ein Review des eigentlichen Quellcodes ergänzt. Fokus wird dabei auf die verwendeten Design-Patterns gelegt.

Am Ende werden die durchgeführte Portierung und ihr Endresultat, AndroidPrefuse, beschrieben. Im Allgemeinen ist die Architektur von AndroidPrefuse nach der Portierung gleich wie prefuse geblieben. Um die erfolgreiche Portierung zu demonstrieren, wurde ein einfaches Streudiagramm in AndroidPrefuse implementiert bzw. portiert. Es werden folgende Fragen in Bezug auf die Portierung beantwortet: Welche Portierungsmöglichkeiten standen zur Verfügung? Warum wurde die Möglichkeit zur Übernahme in AndroidPrefuse von AWT- und Swing-Bibliotheken für die Portierung ausgewählt? Welche Herausforderungen traten bei der Portierung auf, und wie wurden sie gelöst? Welche Teile wurden von prefuse komplett, und welche teilweise bzw. gar nicht übernommen?

Abstract

Information visualization is of great importance in order to allow users to display complex and large amounts of data. It has a long history on the desktop computer, and therefore there are already numerous tools and frameworks that satisfy the needs of the majority of users. Because of the proliferation of mobile devices in recent years and because of the improvement of said mobile devices with regard to performance and screen quality, the demand for information visualization on mobile devices has increased. Since this is a new area, there are relatively few tools and frameworks for this purpose.

The topic of this paper is introduced by displaying the already existing tools and frameworks in Android. Then their functionality and options are briefly explained.

Furthermore, the structure of prefuse, one of the most successful desktop frameworks concerning information visualization, is analyzed. In the course of this analysis, a review of the actual source code is added to the information that has been obtained from the State of the Art on prefuse. Particular attention is paid to used design patterns.

Finally, the conducted porting and its final result, AndroidPrefuse, is described. In general, the architecture of AndroidPrefuse after porting has remained the same as prefuse. To demonstrate the successful porting, a simple scatterplot in AndroidPrefuse has been implemented or rather ported. In addition, the following questions regarding the porting are answered: Which options for the porting were available? Why were the AWT and Swing libraries adopted for AndroidPrefuse? What challenges have been encountered during porting, and how were they accomplished? Which prefuse parts were completely adopted, and which were only partially or not adopted at all?

Inhaltsverzeichnis

1 Einführung

1.1 Forschungsfragen

Die Benutzung von mobilen Geräten hat in den letzten Jahren stark zugenommen [34] (Abbildung 1). Der Marktanteil von Android ist im Vergleich zu anderen Betriebssystemen groß und laut Prognosen von Statista[35] wird sich an dieser starken Verbreitung bis 2018 nicht viel ändern. Zudem sind in den letzten Jahren die Bildschirme mobiler Geräte deutlich größer und leistungsfähiger geworden und deren Performance ist stark gestiegen. Es ist daher naheliegend, vorhandene Visualisierungstechniken für mobile Geräte anzupassen, um sie jederzeit und überall durch Informationsvisualisierungs-Applikationen (kurz InfoVis-Applikationen) nutzen zu können [11].

In Android existieren bereits einige Visualisierungsbibliotheken. Deren Anzahl ist aber kleiner als für den PC. Die meisten und die bekanntesten von ihnen sind nicht kostenpflichtig und bieten eine gewisse Anzahl an fertigen Diagrammen und zudem einige Möglichkeiten für Benutzerinteraktionen. Diese Visualisierungsbibliotheken wurden erstellt, um die oben erwähnten Features (fertige Diagramme und Benutzerinteraktionen) anzubieten und diese eventuell konfigurierbar zu machen. Neue Diagramme, d.h. neue Visualisierungen, können höchstwahrscheinlich nur von den Personen, die in der Community des jeweiligen Projektes beteiligt sind, erstellt werden.

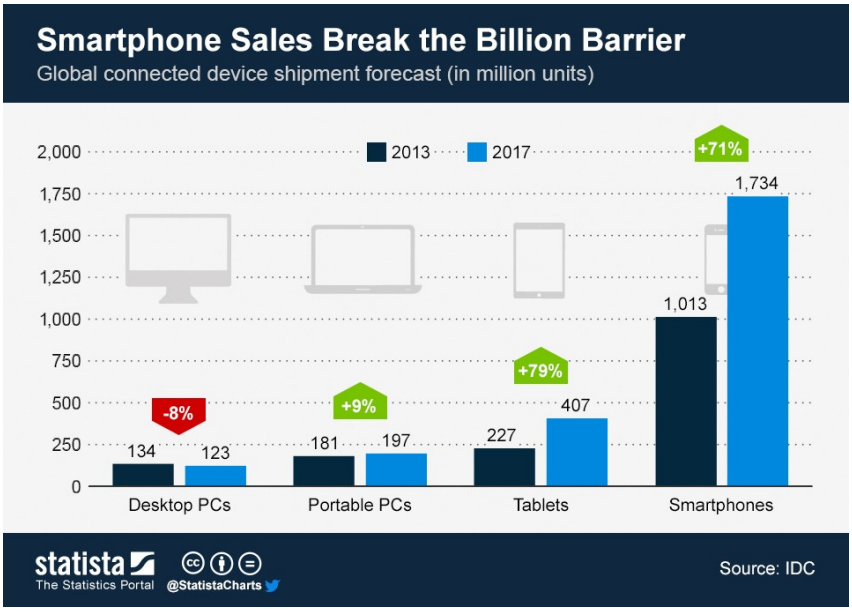

Abbildung 1: Prognose des Verkaufs von mobilen Geräten und Desktop-Rechnern 2013-2017 (Quelle Statista)

Möglicherweise decken bei Android vorhandene Softwarebibliotheken den Bedarf vieler Anwender bereits ab. In der Visualisierungswissenschaft und teilweise auch in der Industrie ist es oft notwendig, für die Visualisierung abstrakter und komplexer Daten spezielle Diagramme/Grafiken zu erstellen. Zusätzlich betonen Heer et al. [22] – Seite 1: „infovis applications do not lend themselves to 'one size fits all' solutions“. Daraus entsteht der Bedarf, ein flexibles und leicht erweiterbares Framework für Visualisierungen zu erstellen.

Für die PC-Landschaft gibt es bereits viele Visualisierungs-Tools bzw. Frameworks. Jedes von ihnen hat seine Stärken und Schwächen. Es gibt viele Tools im High-Level-Bereich, die viele Möglichkeiten anbieten. Allerdings sind sie oft in Bezug auf spezielle Visualisierungen unflexibel. Andererseits gibt es viele und gute Low-Level-Visualisierungstools, die hohe Flexibilität anbieten. Allerdings sind diese Tools in der Regel mühsam und komplex in der Bedienung. Daraus lässt sich schließen, dass es eine Lücke zwischen den High-Level- und den Low-Level-Tools gibt [6].

Diese Lücke versucht prefuse[1], ein Visualisierungswerkzeug für den PC-Bereich, zu schließen. Es ist eine weit verbreitete Programmbibliothek in der Wissenschaft. Es zeichnet sich durch seine Modularität und Flexibilität aus und dadurch versucht, die oben erwähnte Lücke zwischen High-Level- und Low-Level-Visualisierungstools bzw. -frameworks zu schließen [22]. Diese Modularität und Flexibilität wird durch die Anwendung zahlreicher und durchdachter Software-Design-Patterns erreicht. Viele der verwendeten Design-Patterns stammen speziell aus dem prefuse-Fachgebiet der Informationsvisualisierung. Das Hauptdesign-Pattern von prefuse ist das in der Informationsvisualisierung weit verbreitete Architektur-Design-Pattern „Information Visualization Reference Model“. Durch die Anwendung der vielen Design-Patterns ist die schnelle Entwicklung von Visualisierungen mit einem gewissen Einarbeitungsaufwand in prefuse möglich. prefuse ist in Java programmiert und damit in vielen Betriebssystemen verwendbar. Allerdings unterstützt es keine mobilen Plattformen wie Android. Einige darin verwendete Java-Technologien wie Java AWT, Java Swing und Java 2D werden in Android nicht unterstützt [15] und dies macht die direkte Verwendung von prefuse auf Android unmöglich.

Zusammenfassend werden in dieser Arbeit folgende Forschungsfragen bearbeitet:

1. Welche Visialisierungsbibliotheken gibt es bereits unter Android?
2. Welche Software Design-Patterns kommen bei (Desktop) prefuse vor? Sind sie auch für Android geeignet?
3. Welche Vorteile würde ein Visualisierungs-Framework wie prefuse für Android bieten?

---

1 prefuse, abgerufen am 11.12.2014

http://prefuse.org

1.2 Erwartetes Ergebnis

Hauptziel dieser Diplomarbeit ist das Beantworten der gestellten Forschungsfragen aus dem vorherigen Unterkapitel 1.1. Nebenziel ist das Portieren von prefuse nach Android. Die Portierung ist eines der verwendeten Mittel für die Beantwortung der genannten Forschungsfragen. Das Endresultat der Portierung – AndroidPrefuse – soll Grundlagen für dieselben Möglichkeiten und dieselbe Flexibilität wie die Desktop-Variante bieten. Ziel ist vorerst nicht, alle Funktionen von prefuse nach Android zu transferieren, weil dies den Rahmen der Diplomarbeit sprengen würde. Am Schluss soll mindestens ein Streudiagramm durch AndroidPrefuse erstellt werden können. Hauptziel der Portierung ist also, die Modularität und die Software-Architektur von prefuse nach Android zu portieren, sodass es in Zukunft beliebig und problemlos erweitert werden kann. Zusätzlich werden die möglichen Schwachstellen von prefuse durchleuchtet und versucht, diese bei der Portierung zu verbessern.

Am Ende der Arbeit wird klargestellt werden, ob und wie die Portierung von prefuse nach Android durchgeführt werden kann, d.h. es wird ermittelt, welche Visualisierungskonzepte komplett übernommen werden können und welche angepasst werden bzw. neu entwickelt werden müssen. Wenn die Portierung durchgeführt wird, ist das Endresultat AndroidPrefuse, ein Visualisierungs-Framework, das auf prefuse basiert und dieselben Möglichkeiten wie prefuse bietet.

1.3 Übersicht

Als erstes werden in Kapitel 2 „State of the Art“ die wissenschaftlichen Publikationen und Softwarebibliotheken, die verwandt mit dieser Arbeit sind oder eine Relevanz für diese Arbeit haben, beschrieben. In diesem Kapitel findet sich eine kurze Analyse der bekanntesten der vorhandenen Visualisierungsbibliotheken auf Android. Die wissenschaftlichen Datenbanken, die für die Suche bei der Recherche von Informationen verwendet wurden, sind: IEEE Xplore, ACM Digital Library, Springer Link sowie die Suchmaschine „Google Scholar“. Folgende Schlüsselwörter wurden in verschiedenen Kombinationen verwendet:

- android
- graphic
- library
- package
- (information) visualization
- java
- mobile
- prefuse
- framework

Mit den oben aufgelisteten Schlüsselwörtern wurde nichts Relevantes in Bezug auf Visualisierungsbibliotheken in Android gefunden. Daher mussten für die Erstellung der Liste der vorhandenen Android Visualisierungsbibliotheken die Informationen von Webseiten abgerufen werden. Für die Suche im Web (Google) wurden dieselben oben erwähnten Schlüsselwörter verwendet. Google lieferte hierbei sehr viele (über 3,4 Millionen) Ergebnisse. Die meisten Einträge waren auf die Grafikprogrammierung in Android bezogen. Ein großer Teil der Ergebnisse waren Tutorials für die Programmierung von Spielen in Android. Diese wurden für diese Arbeit auch ausgeschlossen, da sie keine Relevanz für Informationsvisualisierung haben. In einem Stackoverflow-Beitrag[2] haben Benutzer/innen die meist relevanten und beliebtesten Android-Visualisierungsbibliotheken aufgelistet.

Das Wissen über Java AWT, Java Swing, Java 2D, Android, Design-Patterns sowie prefuse selbst sind Grundsteine für die Konzeption und Planung der Portierung von prefuse nach Android. In Kapitel 3 „Allgemeine Informationen“ werden die Informationen, die nicht direkt mit dem Thema der Arbeit zu tun haben, sondern allgemeine Grundlagen darstellen, beschrieben. Dabei handelt es sich auch um Informationen, die helfen, die Begriffe und Fachbereiche, die diese Arbeit verwendet und bedient, näher zu erläutern. Wichtige Bereiche, wie Informationsvisualisierung, Design-Patterns, Grafikprogrammierung in Java und Android werden erläutert. prefuse, dessen Architektur und darin verwendete Design-Patterns werden im Detail beschrieben. In diesem Kapitel werden einige der gestellten Forschungsfragen beantwortet.

Das folgende Kapitel 4 „AndroidPrefuse“ ist der Portierung und dem Resultat der Portierung gewidmet. Im Unterkapitel 4.5 werden sowohl technische als auch visualisierungsbedingte Herausforderungen und Versuche für deren Lösungen, die vor bzw. bei der Portierung aufgetreten sind, beschrieben. Ferner wird das Produkt der Portierung und die Implementierung vom Prototyp „Streudiagramm“ genauer erläutert.

Zum Schluss werden die Forschungsfragen in Kapitel 5 „Beantwortung der Forschungsfragen“ zusammengefasst beantwortet. Anschließend werden noch offene Fragen und mögliche zukünftige Arbeiten in Kapitel 6 „Ausblick“ aufgelistet. Am Ende wird die vorliegende Arbeit im letzten Kapitel 7 „Zusammenfassung“ noch einmal rekapituliert.

2 StackOverflow, abgerufen am 15.10.2015

http://stackoverflow.com/questions/6806537/graphs-api-for-android

2 State of the Art

2.1 Allgemeine Visualisierungsbibliotheken

Chittaro [11] beschreibt allgemeine Ansätze, wie und was man bei Visualisierungen von Informationen beachten sollte. Es wird erwähnt, dass eine eins-zu-eins-Portierung von Desktop-Applikationen zu Mobile-Applikationen gar nicht möglich ist. Gründe hierfür sind einerseits die unterschiedlichen Technologien, andererseits die gerätespezifischen Unterschiede, wie z.B. unterschiedliche Hardware, unterschiedliche Bildschirmgrößen, Breiten/Längen-Verhältnis am Bildschirm, unterschiedliche Eingabegeräte, usw.

Ein Ansatz zur Visualisierung von Informationen in mobilen Geräten und damit auch in Android ist das Programmieren von Webbrowser-Applikationen. Lammarsch et al. [26] vergleichen die Technologien dafür, ohne speziell auf mobile Geräte einzugehen. Allerdings lässt sich aus der Veröffentlichung schließen, dass die Technologien, die sich für mobile Geräte eignen, wie z.B. Flash, Probleme mit der Performance haben. Moelker und Wijbrandi [27] verglichen die Performance von einer Applikation, implementiert in einer nativen mobilen Programmiersprache (Objective-C für iOS und Java für Android), mit der einer Web-Applikation, implementiert in WebGL, das als Basis JavaScript hat. Das Ergebnis besagt, dass JavaScript-Applikationen wesentlich langsamer als native mobile Applikationen sind.

prefuse [22] ist eine polylithische Visualisierungsbibliothek für zweidimensionale Grafiken für den Desktop-PC. Es ist in Java implementiert und benutzt auf Java2D basierende Grafikbibliotheken. Es implementiert das Visualisierungs-Design-Pattern „Information Visualization Reference Model“. Basierend auf diesem Architekur-Design-Pattern nimmt prefuse eine Trennung zwischen abstrakten Daten, visuellen Daten, Views und Benutzerinteraktionen vor. Es beinhaltet eine Programmbibliothek von Layout-Algorithmen, Navigations- und Interaktionstechniken, integrierte Suche und vieles mehr.

ThisStar [13] ist eine domänenspezifische Programmiersprache für die Erstellung von Sternkarten. Es beschreibt ein Konzept zum Abstrahieren von Visualisierungsbibliotheken. Im Hintergrund basiert es auf prefuse, betont aber, dass es mit jeder beliebigen Visualisierungsbibliothek umgesetzt werden kann. Es geht zwar nicht auf Mobile-Applikationen ein, aber da es versucht, die Hintergrundtechniken zu abstrahieren, kann man daraus schließen, dass es auch bei Mobile-Applikationen Anwendung finden kann. Dafür wird aber eine gute Visualisierungsbibliothek in mobilen Umgebungen benötigt bzw. vorausgesetzt, da es darauf basiert.

PRISMA Mobile [24] ist ein Visualisierungstool, das für Android-Tablets entwickelt worden ist. Die Architektur des Tools basiert auf einem Client-Server-Model. Als Server dient PRISMA [19], das Desktop-Tool mit demselben Name. Dieses kann auf dem Desktop auch eigenständig laufen. Das Visualisierungs-Tool PRISMA basiert auf

„multiple coordinated views“ [19] um folgende Visualisierungen zu unterstützen: Treemap, Streudiagramme und parallele Koordinaten. Die Client-Server-Architektur ist dafür gedacht, die Leistung und Batterielaufzeit des Tablets zu schonen, weil diese im Gegensatz zum Server stark begrenzt ist.

Diese Architektur kann große Vorteile mit sich bringen, wenn für die Erstellung der Grafiken viel Rechenleistung gebraucht wird, wie z.B. beim Berechnen von Grafiken mit sehr vielen Rohdaten. Allerdings bringt diese Architektur auch einige Nachteile mit sich: als Erstes die Notwendigkeit des Aufsetzens eines Servers, was nicht immer einfach ist. Es ist schwer abschätzbar, ab wann dieser Client-Server sich lohnt, denn das Ergebnis ist auch von der Bandbreite des Netzwerkes abhängig. Ein schwaches Netzwerk bzw. ein großer Netzwerkverkehr kann sich auf die Leistung des Tools negativ auswirken. Die Leistungsfähigkeit der aktuellen Tablets und Smartphones steigt stetig und dies könnte in naher Zukunft die Client-Server-Architektur für die meisten Anwendungsgebiete überflüssig machen. Außerdem ist dies ein fertiges Tool und keine Softwarebibliothek bzw. kein Software-Framework. Das bedeutet, dass nur die obenerwähnten Visualisierungen unterstützt werden. Für weitere bzw. individuelle Visualisierungen ist wahrscheinlich mit hohem Aufwand zu rechen, wenn es überhaupt als Dritter möglich ist, die Software zu erweitern. Aus dem Paper war nicht ersichtlich, ob es sich bei der Software um Open Source handelt. Zusätzlich müssen bei Erweiterungen Anpassungen sowohl beim Client als auch beim Server vorgenommen werden.

TimeBench [32] ist eine Softwarebibliothek für Visual Analytics, die auf prefuse basiert. Es liefert grundlegende Datenstrukturen und Algorithmen für zeitbezogene Daten im Visual Analytics-Bereich. Damit ist es möglich, mit relativ wenig Aufwand zeitbezogene Visualisierungen aufzubauen.

Bostock und Heer [6] haben einen anderen interessanten Ansatz. Sie haben eine domänenspezifische Sprache (engl. domain-specific language; kurz DSL) namens **Protovis** für die Informationsvisualisierung erstellt. Die Sprache basiert auf JavaScript. In der Veröffentlichung wird gar nicht auf die Anwendung bei mobilen Geräten eingegangen. Da die Sprache auf JavaScript basiert, sind höchstwahrscheinlich die Applikationen, die damit gebaut sind, weniger performant als native mobile Applikationen, basierend auf [10], [27]. Laut der Protovis Webseite[3] wird das Produkt nicht mehr weiterentwickelt.

D3 [7] ist ein Nachfolger von Protovis [6]. Es basiert auf vielen Protovis-Konzepten, verwendet JavaScript als Programmiersprache und ist für das Zeichnen im Web-Browser gedacht. Für die Datendarstellung wird versucht, die vorhandenen Techniken im Browser so viel wie möglich zu benutzen. D3 speichert die Daten als Elemente im DOM (document object model) und zeichnet die Grafiken mittels Browser-Vektorgrafik SVG. Dies hat laut Autor/innen Vorteile, wie z.B. Browser-Kompatibilität und leichte Lernbarkeit des

3 Protovis, abgerufen am 11.12.2014
http://mbostock.github.io/protovis/

Tools wegen Benutzung von vorhandenen Standards. Laut einem Benchmark [7] zwischen D3, Protovis und Flash Grafiken, hat D3 die beste Seitenladezeit. Allerdings ist Flash bei Bildfrequenz (frame rate) wesentlich performanter (2 bis 2,5 Mal höhere Bildfrequenz) als D3 und Protovis. Dies zeigt, dass webbasierte Applikationen für Informationsvisualisierung immer noch Probleme mit der Performance haben.

SVGAnnotation [29] ist ein weiterer Ansatz für das Zeichnen von Grafiken auf dem Browser. Das Tool basiert auf R-Grafik [36] und Browser-Vektorgrafik SVG. Die Grafiken werden im R-Grafik-Engine erzeugt, und als Ergebnis wird eine SVG Grafik erstellt. In die erzeugte Grafik können dann mittels JavaScript Benutzerinteraktionen eingebaut werden. Die R-Grafik-Engine ist zur Laufzeit (während Benutzerinteraktionen) nicht mehr erreichbar. Für die Manipulierung der Grafik muss man dann zu anderen Mitteln (reines JavaScript oder andere Tools) greifen.

2.2 Spezielle Visualisierungsbibliotheken (Android)

Da die Android-Plattform eine kürzere Geschichte als Desktop-Plattformen hat, ist die Anzahl der Visualisierungsbibliotheken in Android im Vergleich zu den Desktop-Applikationen geringer. Die meisten von ihnen sind Open Source und folgende sind die bekanntesten:

AchartEngine:[4] unterstützt 12 Arten von Diagrammen. Bietet keine Benutzerinteraktion an. Das letzte Release der Stable-Version 1.1.0 der Software war am 15.05.2013. In den Abbildungen 3 und 2 werden zwei der möglichen Diagramme von AchartEngine abgebildet.

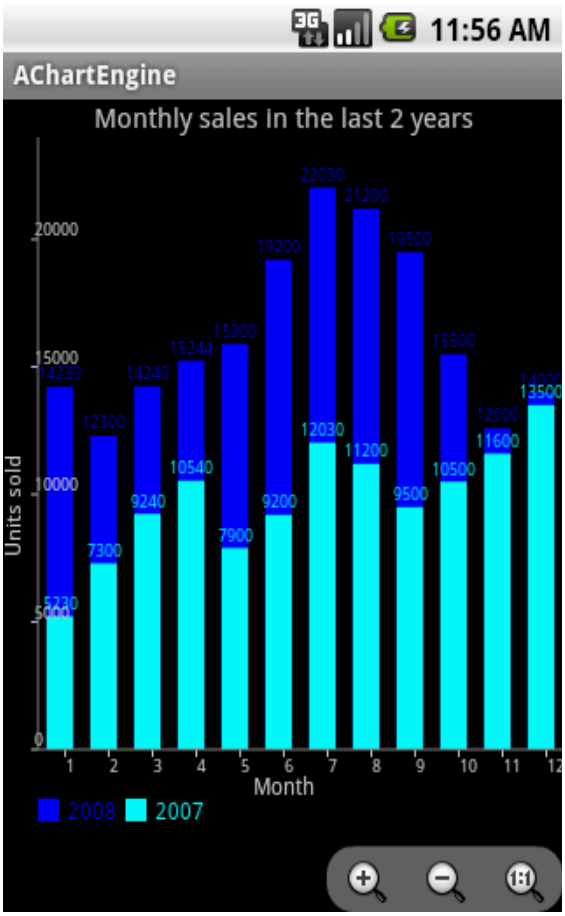

Abbildung 3: AChartEngine – Säulendiagramm (Quelle: Webseite von AChartEngine)

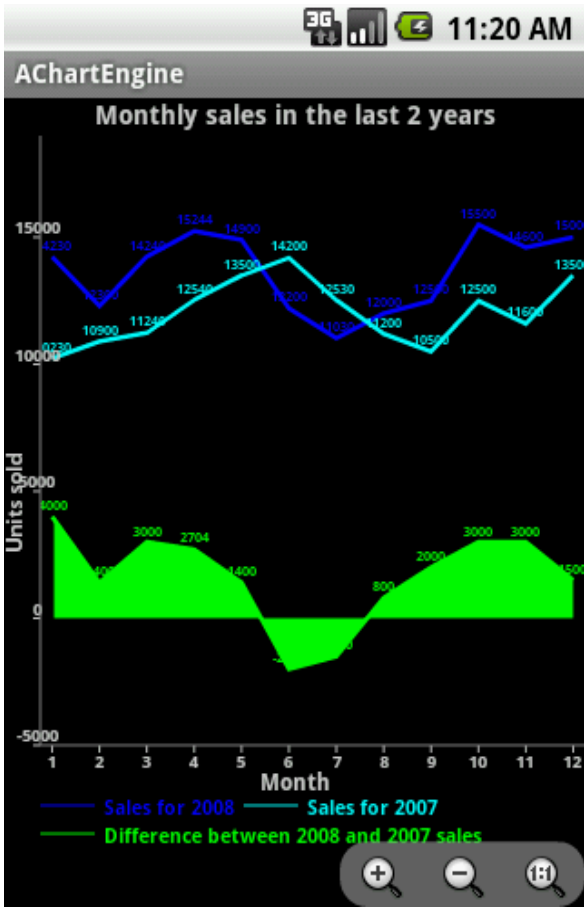

Abbildung 2: AChartEngine - Ein Linien-zusammen mit einem Flächendiagramm

(Quelle: Webseite von AChartEngine)

4 Achartengine, abgerufen am 11.12.2014
http://www.achartengine.org/

Androidplot:[5] unterstützt fünf Arten von Diagrammen. Als Benutzerinteraktion bietet es Verschieben, Zoom und Skalieren. Die aktuellste Stable-Version ist 0.6.1. Auf der Webseite des Tools gibt es reichlich Anleitungen, wie man vorhandene Diagramme in eigene Android-Applikationen einbauen bzw. konfigurieren kann. In Abbildung 4 werden verschiedene Diagramme zusammen mit den Rohdaten in einem Bildschirm angezeigt. In Abbildung 5 wird die Möglichkeit dargestellt, die Diagramme als Widgets auf dem Desktop anzuzeigen.

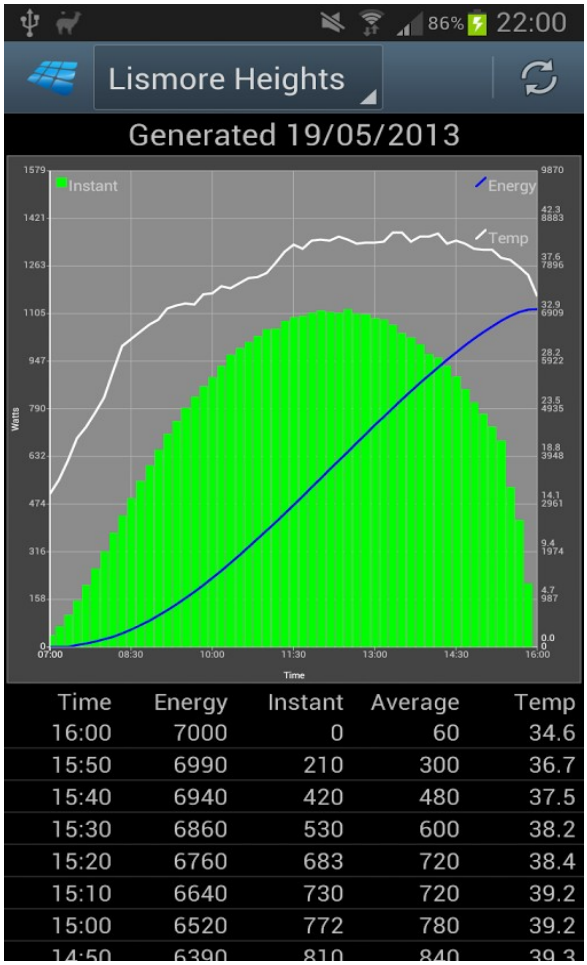

Abbildung 4: Androidplot - Verschiedene Diagramme zusammen mit den Rohdaten in einem Bildschirm (Quelle: Webseite von Androitplot)

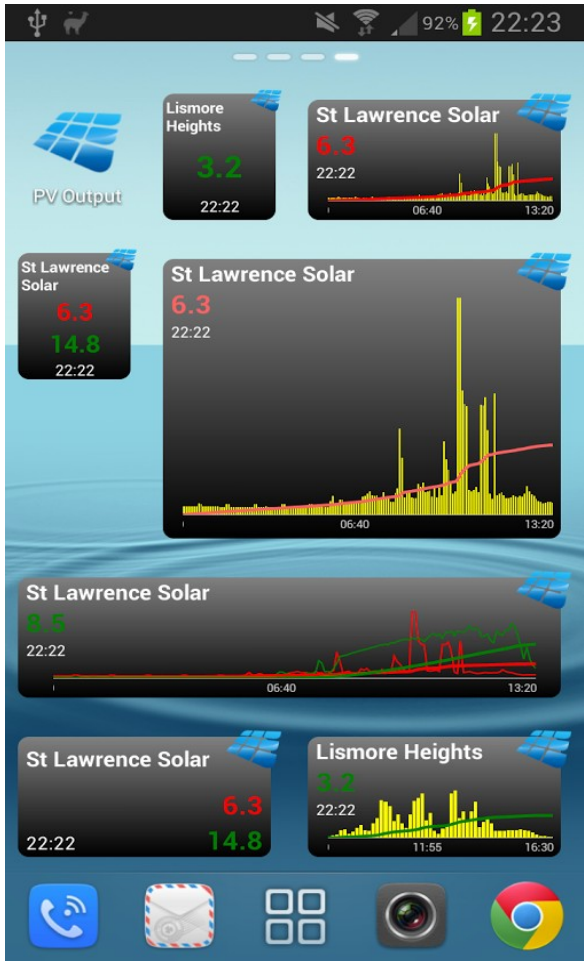

Abbildung 5: Androidplot - Die Diagramme, dargestellt als Widgets auf dem Desktop (Quelle: Webseite von Androitplot)

5 Androidplot, abgerufen am 11.12.2014
http://androidplot.com/

GraphView:[6] unterstützt zwei Arten von Diagrammen. Als Benutzerinteraktion bietet es Zoom und Skalieren. Die letzte Stable Release Version 4.0 wurde am 09.12.2014 veröffentlicht. Bei einer schnellen Untersuchung des Quellcodes wurde festgestellt, dass dieser sehr wenige Klassen beinhaltet. Dies deutet daraufhin, dass kein Wert auf Modularität gelegt wurde. In Folge dessen ist die Erweiterung dieses Tools mit neuen Diagrammen sehr aufwendig. In Abbildung 6 wird ein einfaches Punktediagramm abgebildet.

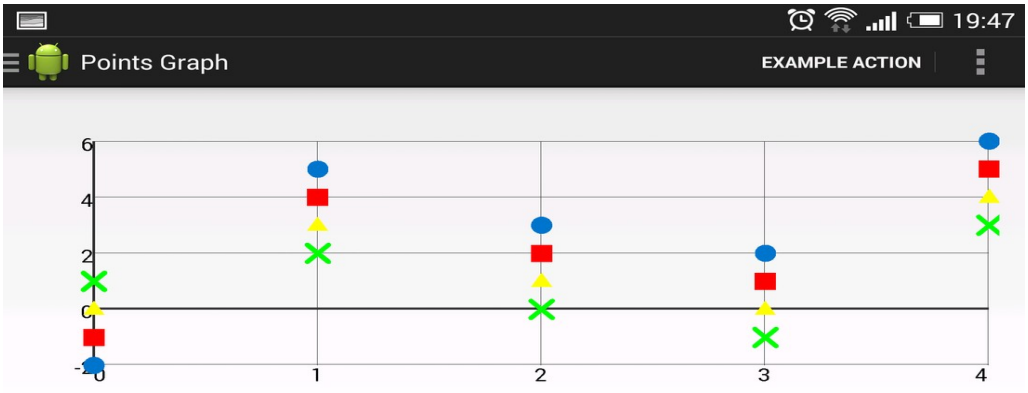

Abbildung 6: Graphview – Punktediagramm (Quelle Webseite von Graphview)

HoloGraphLibrary:[7] bietet drei Arten von Diagrammen an und unterstützt keine Benutzerinteraktionen. In den Abbildungen 7 und 8 werden zwei der möglichen Diagramme abgebildet. Auf der Webseite gibt es keine Informationen über ein Release-Programm. Das Tool wird anscheinend direkt aus dem git-Reporistory heruntergeladen. Dies zeigt, dass das Tool noch nicht ausgereift ist. Wie bei GraphView bemerkt, hat dieses Tool wenige Java-Klassen, was darauf hindeutet, dass eine Erweiterung des Tools mit viel Aufwand verbunden ist.

6 Android GraphView, abgerufen am 11.12.2014
http://android-graphview.org/

7 Nadeau, D. Holographlibrary, abgerufen am 11.12.2014
https://bitbucket.org/danielnadeau/holographlibrary/wiki/Home

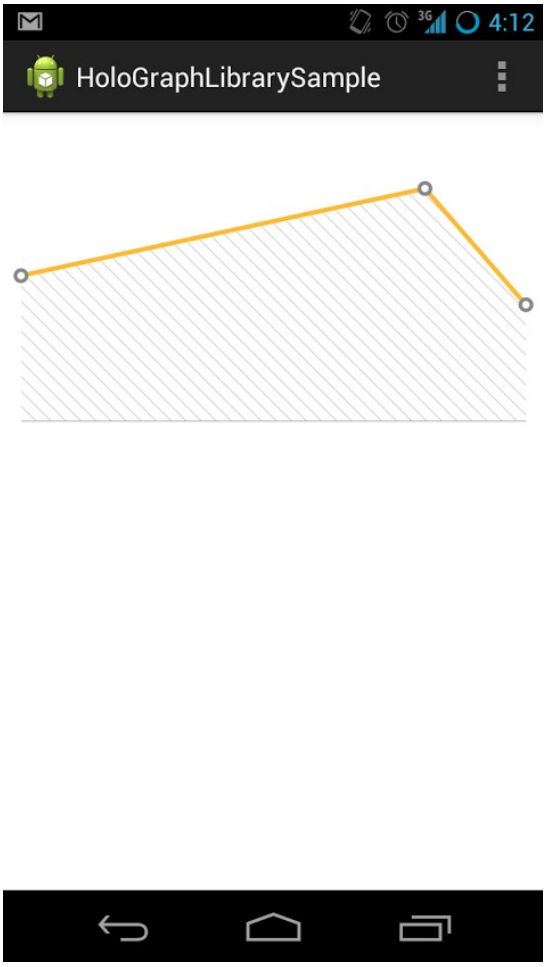

Abbildung 7: HoloGraphLibrary - Liniendiagramm (Quelle Webseite von HoloGraphLibrary)

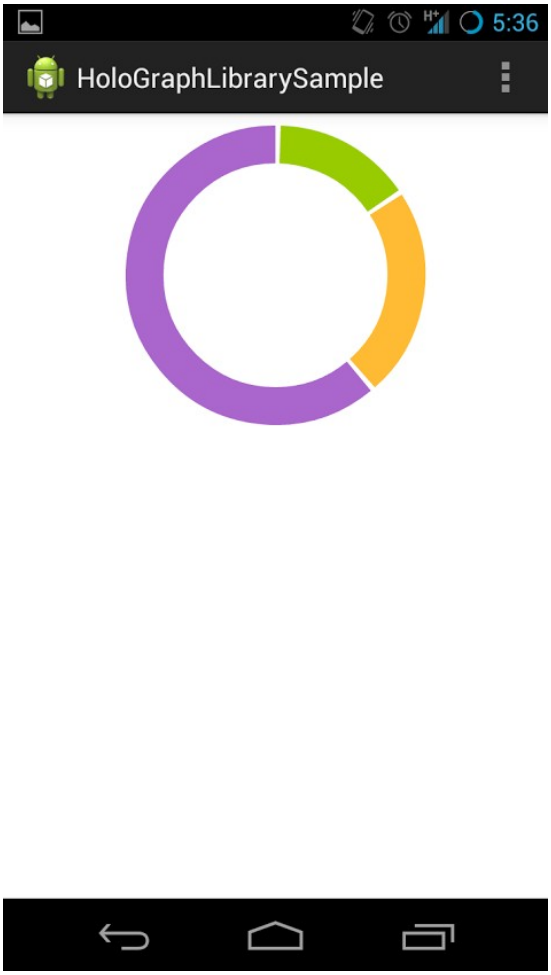

Abbildung 8: HoloGraphLibrary - Kreisdiagram (Quelle HoloGraphLibrary Webseite)

Charts4j[8]: ist eine Java-Implementierung von Google Chart Tools (siehe Kapitel 3.5.3). Da es auf keine betriebssystembedingten grafischen Bibliotheken setzt, kann es in allen Umgebungen (inkl. Android), in denen Java laufen kann, verwendet werden. Es bietet ähnliche Diagramme wie Google Chart Tools, unter anderem Säulendiagramme, Streudiagramme, Liniendiagramme, Flächendiagramme, Netzdiagramme, Kreisdiagramme, Kartendarstellung, usw. Da es im Hintergrund die Cloud-API von Google benutzt, muss eine permanente Internetverbindung vorhanden sein, damit die Diagramme gezeichnet werden können. In Abbildung 9 ist ein Liniendiagramm, das mit dieser Bibliothek erstellt wurde, dargestellt.

8 charts4j – Visualisierungsbibliothek, abgerufen am 15.10.2015
https://github.com/julienchastang/charts4j

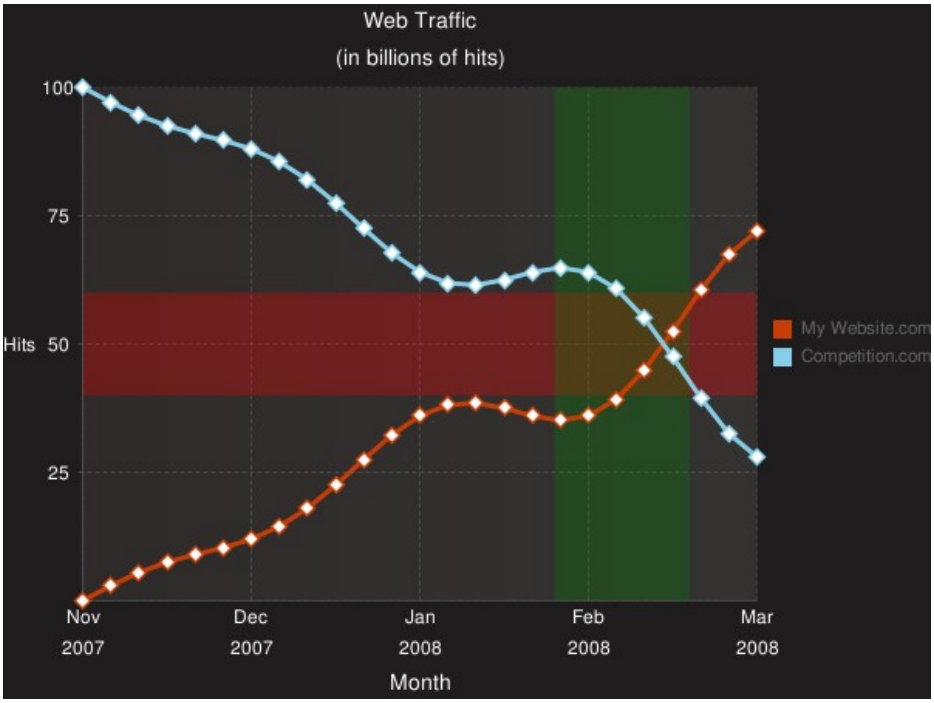

Abbildung 9: Liniendiagramm mit charts4j (Quelle: Webseite von charts4j)

Bei allen oben aufgelisteten Tools fällt auf, dass es wenig bzw. keine Dokumentation (außer der API) in Bezug auf die Architektur des Tools gibt. Deswegen kann man keine klaren Aussagen zur Architektur und in Folge dessen zur Erweiterbarkeit bzw. Anpassbarkeit des Tools machen.

In diesem Abschnitt wurden einige Tools, die Grafiken im Browser mittels JavaScript erzeugen, vorgestellt. Sie wurden hauptsächlich wegen der im Vergleich zu nativen Programmiersprachen in der jeweiligen Plattform schlechten Performance kritisiert. Der Ansatz, Grafiken im Browser anzuzeigen, hat aber viele Vorteile. Die komplette Plattformunabhängigkeit ist dabei das größte Pro. Die Entwickler/innen müssen nur den Code für diese eine Plattform schreiben. Damit werden mehr Benutzer/innen von der Software angesprochen. Die Wartbarkeit der Applikation ist natürlich wesentlich leichter, da der Code nur in einer Plattform gewartet werden muss. Zusätzlich arbeiten die Browserhersteller/innen ständig daran, die Performance in Bezug auf Grafiken zu steigern. Wenn dieser Trend anhält, werden diese Ansätze keine wesentlichen Nachteile mit sich bringen.

In diesem Kapitel wurden wissenschaftliche Publikationen, die im selben oder einem ähnlichen wissenschaftlichen Umfeld wie diese Arbeit stehen, aufgelistet. Damit wurde die erste Forschungsfrage (siehe „Forschungsfragen„) aus wissenschaftlicher Sicht beantwortet. Da diese Frage sich aus den wissenschaftlichen Publikationen nicht direkt und komplett beantworten ließ, wurde nach Visualisierungsbibliotheken im Web gesucht. Um den Bekanntheitsgrad zu ermitteln, wurden die Informationen entweder direkt den

Webseiten der Bibliotheken oder den Benutzer-Foren entnommen. Es konnte also keine belegte Statistik gefunden werden, in der die bekanntesten Android-Visualisierungsbibliotheken aufgeführt sind. Aus der Websuche entstand eine Liste (Tabelle 1) mit den bekanntesten Android-Visualisierungsbibliotheken.

Bibliothek	*Version/Datum*	*Anzahl an Visualisierungen*	*Benutzer-interaktion*	*Notizen zu Erstellung von neuen Visualisierungen*
AchartEngine	1.1.0 / 15.05.2013	12	Nein	Über die Möglichkeit für die Erstellung von neuen Visualisierungen kann man wegen mangelhafter Dokumentation keine klaren Aussagen machen.
AndroidPlot	0.6.1 / 01.07.2013	5	Ja	Über die Möglichkeit für die Erstellung von neuen Visualisierungen kann man wegen mangelhafter Dokumentation keine klaren Aussagen machen.
GraphView	3.1.4 / 02.11.2013	2	Ja	Erstellung von neuen Visualisierungen kann schwierig sein. Am Quellcode ist ersichtlich, dass eine Visualisierung in einer Klasse verpackt ist. Dies erschwert die Wiederverwendung von Komponenten.
HoloGraphLibrary	- / 19.10.2014	3	Nein	Erstellung von neuen Visualisierungen kann schwierig sein. Am Quellcode ist ersichtlich, dass eine Visualisierung in einer Klasse verpackt ist. Dies erschwert die Wiederverwendung von Komponenten.
charts4j	1.3 / 01.2011	>7	Nein	Da das Programm komplett auf Google Chart Tools setzt, ist es auf die Diagramme beschränkt, die Google anbietet. Die Erstellung neuer Diagramme, die Google nicht anbietet, ist damit nicht möglich. Allerdings sind bei charts4j nicht alle von Google angebotenen Diagramme implementiert.
prefuse	- / 29.04.2014	> 9	Ja	prefuse ist speziell dafür konzipiert worden, neue Arten von Visualisierungen zu erstellen. Es werden neun Beispiele mit unterschiedlichen Visualisierungen mitgeliefert. Einfache Visualisierungen wie z.B. Streudiagramme lassen sich mit wenig Code erstellen (siehe Anleitung von Alexander Rind[31]).

Tabelle 1: Vergleich zwischen Android Visialisierungsbibliotheken und Prefuse Desktop

3 Allgemeine Informationen

3.1 Begriffe

Um Klarheit bei der Terminologie in dieser Arbeit zu schaffen, möchte ich einige Begriffe näher erläutern.

Tool ist eine Software-Applikation, die bestimmte Funktionen für eine oder mehrere Aufgaben in eine oder unterschiedliche Gebiete bereitstellt. Es entspricht der genauen deutschen Übersetzung, dem Werkzeug. Je nachdem, wie umfangreich das Tool ist, kann man damit unterschiedliche Aufgaben erledigen. Einige Tools bieten bewusst wenige Funktionen und Einstellmöglichkeiten, damit unerfahrene Benutzer/innen, die die breite Masse darstellen, angesprochen werden können. Andere Tools bieten mehrere Funktionen und Einstellmöglichkeiten. Damit wollen sie erfahrene Benutzer/innen in einem bestimmten Gebiet ansprechen. Ein triviales Beispiel für einfache und komplexe Einstellmöglichkeiten ist die Camera-Applikation in Smartphones. Es gibt einige Tools, die wenige Einstellmöglichkeiten anbieten, und es gibt einige, die eine große Varianz an Optionen anbieten. Die breite Masse möchte mit einem Klick/Touch Bilder machen und interessiert sich nicht für die vielen Einstellmöglichkeiten. Erfahrene Benutzer/innen im Bereich der Fotografie wollen pro Schnappschuss die Möglichkeit haben, Optionen zu verändern, um bessere und schärfere Bilder schießen zu können.

Ein **Framework** ist wie das Tool ein Stück Software. Im Gegensatz zu Tools ist dessen Aufgabe nicht, fertige Funktionen anzubieten, sondern ein Grundgerüst für das Bauen einer Software-Applikation bereitzustellen. Dabei kann es auch vorkommen, dass es fertige Funktionen anbietet, diese sind aber in der Regel kleine Funktionen mit vielen Einstellmöglichkeiten (aus Software-Sicht), die man miteinander verbinden kann, um damit komplexere Funktionen zu erstellen. Diese können auch erweitert werden. Gänzlich neue Funkionen können erstellt werden, aber in der Regel nur unter den jeweiligen Voraussetzungen, die das Framework vorgibt. Diese Voraussetzungen sind meistens technisch und nicht durch Richtlinien vom Framework bestimmt. Frameworks sind also für Softwareentwickler/innen gedacht und nicht wie Tools für den Endbenutzer/die Endbenutzerin. Durch das Bereitstellen der vielen kleinen Funktionen wird dem Entwickler/der Entwicklerin die Arbeit erleichtert, indem er/sie diese Funktionen nicht neu entwickeln muss. Durch das Bereitstellen der Voraussetzungen/Einschränkungen wird das Grundgerüst der Applikation bestimmt. Der Hauptgrund für dieses Grundgerüst ist, dass die Applikation für bestimmte Gebiete eine bestmögliche Basis und damit eine bewährte Architektur hat. Ferner sind Frameworks dazu gedacht, Applikationen leichter wartbar und erweiterbar zu machen. Wartbarkeit und Erweiterbarkeit sind die Hauptgründe für die Gestaltung von Frameworks. Manchmal stellen sie auch ein Gerüst für eine optimierte Performance der Applikation. Eine zusätzliche Aufgabe der Frameworks ist es, dem/der Programmierer/in die Möglichkeit zu geben, eine Software-Applikation in kurzer

Zeit fertigzustellen. In der Regel ist das Schreiben einer Software-Applikation ohne ein Framework mehr Aufwand und Arbeit.

Eine **Softwarebibliothek** ist nur eine Zusammenstellung vieler kleiner und granulärer Software-Funktionen für ein bestimmtes Gebiet oder eine bestimmte Aufgabe. Im Gegensatz zum Framework bietet sie kein Grundgerüst für das Bauen komplexerer Applikationen. Softwarebibliotheken sind wie Frameworks für Softwareentwickler/innen und nicht für Endbenutzer/innen bestimmt.

3.2 Design-Patterns

Design-Patterns sind Lösungsschablonen für Entwurfsprobleme in der Softwareentwicklung. Sie bieten Vorlagen für wiederkehrende Probleme aus der Softwarearchitektur und -entwicklung in einem bestimmten Kontext. Diese Lösungsschablonen haben sich mit der Zeit als bewährte Lösungen für bestimmte Probleme entwickelt. Sie sind darauf ausgelegt, die Wartbarkeit, Wiederverwendung und andere Aspekte der Softwareentwicklung zu unterstützen.

Gamma et al. beschreiben Design-Patterns wie folgt:

> „The design patterns in this book are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.“ [17] – Seite 13

3.3 Grafikprogrammierung in Java

Das Programmieren von grafischen Benutzeroberflächen in Java wird durch Java Foundation Classes (JFC) bewerkstelligt, die eine Sammlung von Programmierschnittstellen (APIs) darstellen. Zu JFC gehören folgende Programmierschnittstellen:

- **Abstract Window Toolkit (AWT)** – Basisklassen für GUI-Komponenten, Events und Layouts
- **Swing** – Erweiterung von AWT durch Drag & Drops-Möglichkeiten, Look and Feel, usw.
- **Java 2D –** Zeichnen von 2D-Elementen
- **Java Accessibility API (JAAPI)** – Programmierschnittstelle für barrierefreie Java-Anwendungen
- **Internationalization** – Unterstützung für Eingabe und Ausgabe von Texten in unterschiedlichen Sprachen.

Die Grundklasse in JFC ist die abstrakte Klasse *java.awt.Component*, die auch die Oberklasse vieler Komponenten darstellt. Beispiele für Komponenten sind: Applets, Knöpfe, Checkbox, Scrollbar, Panels, usw. Die Klasse *Component* stellt Basismethoden für alle

AWT- und Swing-Komponenten zur Verfügung. Eine dieser Basismethoden ist die Methode *paint*, die jedes Mal, wenn die Komponente neu am Bildschirm gezeichnet werden soll, automatisch aufgerufen wird. Die *paint*-Methode ist standardmäßig leer, und wenn man etwas Bestimmtes auf eine Komponente zeichnen will, muss man diese Methode überschreiben. In dieser Methode wird als Parameter ein grafischer Kontext (die Klasse *java.awt.Graphics* oder abgeleitete Klassen davon) übergeben. Auf diesem werden die meisten Operationen, wie z.B. *drawString*, *drawLine*, *setColor*, usw. durchgeführt.

3.3.1 Java AWT

AWT bietet grundlegende grafische Operationen an. Alle Klassen und Methoden dieser API sind im Hauptpaket *java.awt* vorhanden.

Folgende Funktionen bietet AWT an:

- Primitive Grundfunktionen zum Zeichnen von Linien, Flächen und zur Ausgabe von Text
- Methoden zur Steuerung des Programmablaufes über Tastatur-, Maus- und Fensterereignisse (z.B. was soll bei einem Mausklick in einem bestimmten Bereich der Komponente passieren oder was soll passieren, wenn das Programm-Fenster minimalisiert wird, etc.)
- Dialogelemente (z.B. PopUp-Fenster) zur Kommunikation mit dem Anwender
- Funktionen zur Ausgabe und Manipulation von Bitmaps und Tönen

Die AWT-Komponenten übergeben die auszuführenden Aktionen (z.B. Zeichnen am Bildschirm) an plattformspezifische GUI-Objekte. Diese Objekte haben auf unterschiedlichen Betriebssystemen unterschiedliche Darstellung. Deswegen werden im AWT nur Funktionen zur Verfügung gestellt, die auf dem jeweiligen Betriebssystem bereitgestellt werden. Dafür haben Applikationen, die mit AWT erstellt wurden, gleiches Design wie native Applikationen im jeweiligen Betriebssystem.

3.3.2 Java Swing

Die Swing-Bibliothek ersetzt und erweitert die Komponenten- und Behälterklassen des AWTs. Im Gegenzug zum AWT sind die Swing-Komponenten fast alle vollständig in Java geschrieben. Daher sind Form und Funktion unabhängig vom Betriebssystem. Über das sogenannte „Pluggable Look and Feel“ kann die Oberfläche zur Laufzeit geändert und an das jeweilige Betriebssystem angepasst werden. Zusätzlich zum Look and Feel bietet Swing unter anderem folgende Funktionen an: Drag & Drop, Tastenkombinationen zur Steuerung von Komponenten, Tooltips, usw.

3.3.3 Java 2D

Java 2D dient der Darstellung und Modifikation zweidimensionaler Objekte. Diese API ermöglicht sowohl primitive Zeichenoperationen wie das Zeichnen von Linien, Kreisen, Rechtecken etc., als auch komplexe Operationen wie Transformierung, Schneiden, Kombinierung von Formen, Texten und Bildern. Alle Operationen werden von der Klasse Graphics2D, die der *paint*-Methode der jeweiligen Komponente übergeben wird, zur Verfügung gestellt.

3.4 Android

Android ist ein Betriebssystem für mobile Geräte wie z.B. Smartphones, Tablets, Netbooks, usw. Mittlerweile wurde Android oder Derivate davon weiterentwickelt, damit es auf Smartwatches und anderen kleinen Geräten, sogenannten Wearables, laufen kann. Android wird von Open Handset Alliance, das von Google gegründet wurde, entwickelt. Es basiert auf einem Linux-Kernel als Hardware Abstrahierungsschicht. Damit werden Hardware-Komponenten wie Treiber, Speicherverwaltung und Netzwerk unterstützt. Darüber ist eine Schicht in der Programmiersprache C/C++ geschrieben, die Software-bibliotheken wie OpenGL ES, SQLite, Webkit, usw. anbietet. In einer noch höheren Schicht liegt Dalvik VM (bis Android 4.4) bzw. ART – Android Runtime (ab Android 4.4). Dies sind virtuelle Ausführungsumgebungen (virtuelle Maschinen), die die Software in ihrem eigenen Bytecode ausführen. Android benutzt also nicht die Java Virtual Machine, sondern eine eigene Laufzeitumgebung. Daher kann Java Bytecode nicht auf Android laufen. In Android wird der Java-Code in einen Bytecode umgewandelt, der von Dalvik bzw. ARM verstanden wird und sich stark vom Java Bytecode von Oracle unterscheidet, bevor es auf dem Android-Gerät läuft. Die Android-Laufzeitumgebungen benutzen eine Teilmenge von Apache Harmony[9] für die Implementierung der Java Kernklassen, wobei sie sich nicht an JAVA SE und Java ME orientieren. Daher unterstützen sie viele Bibliotheken von Java SE und Java ME nicht. Zu den nicht unterstützen Klassen gehören die J2ME Klassen, ATW und Swing [34]. Weil der Bytecode von Dalvik bzw. von ARM kein Java Bytecode ist, ist das Laden von jar-Bibliotheken, die einen Java Bytecode beinhalten, nicht möglich. Für das Laden von jar-Bibliotheken ist ein spezieller Vorgang notwendig.

Auf den oberen Schichten von Android liegt die in Java geschriebene Software. Als erstes kommt der Application Framework, bestehend aus Applikationsdiensten wie Activity Manager, Resource Manager, View System, usw. Auf der obersten Schicht liegen die Java Software Applikationen (sogenannte Apps). In Abbildung 10 ist die gesamte Android-Struktur dargestellt.

---

9 Apache Harmony abgerufen am 19.04.2015

http://harmony.apache.org/

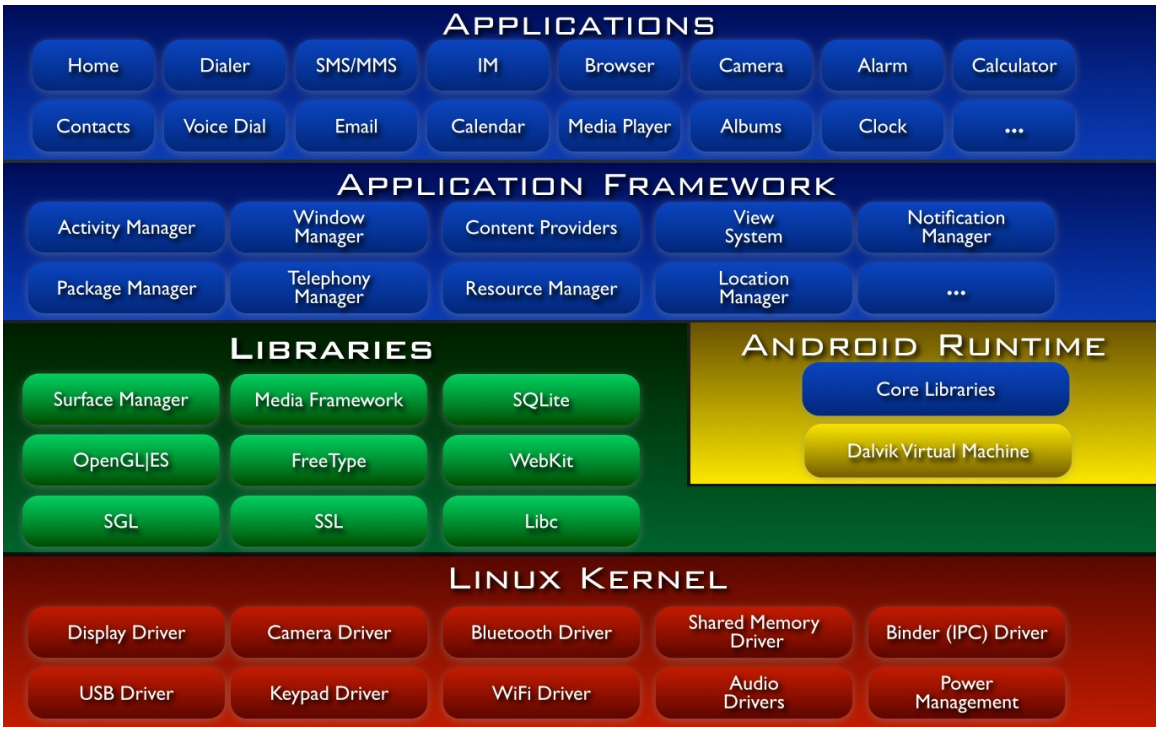

Abbildung 10: Android Anatomy (Quelle [8])

3.4.1 Programmieren in Android

3.4.1.1 Grafikprogrammierung in Android

Die Zeichenfunktion in Android stellt die Klasse *Canvas* zur Verfügung. *Canvas* selbst wird innerhalb der *View*-Klasse bereitgestellt, der Oberklasse aller UI-Elemente. Die *Views* sind einer *Activity* zugeordnet. Bei einer Android-Applikation stellt eine Seite auf dem Bildschirm eine *Activity* dar. Diese Struktur ist in Abbildung 11 zu sehen.

Abbildung 11: Grobe Klassenstruktur in Android

Wie jede Plattform bietet auch Android eine Reihe an primitiven Zeichenoperationen über die Canvas-Klasse, darunter z.B. *drawLine()*, *drawOval(), drawRect(), drawArc(),* etc. Im Großen und Ganzen bietet Android dieselben Möglichkeiten an, die Java mit Graphics2D für das Desktop zur Verfügung stellt. Da prefuse keine Softwarebibliotheken von Drittanbietern, sondern nur die Standard-Java-Funktionen verwendet, sind die Grundvoraussetzungen für das Portieren von prefuse nach Android durch die Standard-Android-Funktionen erfüllt.

Ein ganz anderer Weg zum Zeichnen von Grafiken in Android kann mit Hilfe der Android-Klasse **GLSurfaceView** gegangen werden. Mit ihr ist es möglich, mit OpenGL ES[10] grafische Applikationen zu programmieren. OpenGL ES ist der Standard für plattformübergreifende beschleunigte 2D- und 3D-Grafiken in eingebetteten Systemen. Android unterstützt OpenGL ES Version 2 ab Android 2.2, Version 3 ab Android 4.3 und Version 3.1 ab Android 5[11].

Das Framework **libGDX**[12] ist eine plattformübergreifende OpenGL ES Bibliothek. Es unterstützt Java-Desktop-Plattformen, Android, BlackBerry, iOS und HTML5 für Browser-Anwendungen. Der Quellcode des Frameworks ist frei verfügbar, und laut der Webseiteninformationen wird dieses Framework schon bei vielen Anwendungen (momentan 1745) verwendet. Ahmed und Aule [2] erläutern ihre Erfahrungen mit libGDX und listen auch die Vorteile der Nutzung des Frameworks auf.

OpenGL ES und libGDX zielen auf die Entwicklung von Spielen oder 3D-lastige Applikationen ab, da sie über eine komplexere Logik verfügen und dadurch grafikintensiver sind. Für die Entwicklung von InfoVis-Applikationen sind sie etwas überdimensioniert. Deswegen lohnt sich der Einarbeitungsaufwand meistens nicht.

3.5 Informationsvisualisierung

3.5.1 Allgemeines über Informationsvisualisierung

Informationsvisualisierung ist die Vorbereitung, Verarbeitung, Interpretation und schlussendlich Darstellung in visueller Form von Daten oder einem Teil davon. Heer et al[22] geben die folgende Beschreibung für Informationsvisualisierung „Information visualization (or infovis) seeks to augment human cognition by leveraging human visual capabilities to make sense of abstract information,“[9] „providing means by which humans with constant perceptual abilities can grapple with increasing hordes of

10 OpenGL ES, abgerufen am 11.12.2014

https://www.khronos.org/opengles/

11 OpenGL ES Android, abgerufen am 11.12.2014

http://developer.android.com/guide/topics/graphics/opengl.html

12 LibGDX, abgerufen am 11.12.2014

http://libgdx.badlogicgames.com/

data.“[22], Seite 1. Heer et al haben dabei Card et al zitiert, wobei das Original wie folgendes lautet:

> „The use of computer-supported, interactive, visual representations of abstract data to amplify cognition“ [9], Seite 6

In Abbildung 31 ist das Referenzmodell für den Informationsvisualisierungsprozess dargestellt. In Kapitel 3.6.2.1 wird dieses Design Pattern genauer erklärt.

3.5.2 Ausgewählte Arten von Visualisierungen

Viele Visualisierungsarten sind die Grafiken aus dem Bereich Statistik, die aber auch für InfoVis adaptiert wurden. Die bekanntesten sind Linien-, Balken-, Punkt- und Kreisdiagramme. In den letzten Jahren wurden viele Visualisierungen entwickelt, die oft komplex und speziell für ein bestimmtes Gebiet oder einen konkreten Anwendungsfall gedacht sind. Da die Liste der Visualisierungen sehr lang ist, werden im Folgenden nur die meist verwendeten Visualisierungen kurz umrissen. Die Frage, ob die unten vorgestellten Visualisierungen sich für den Einsatz in mobilen Geräten eignen, ist ein eigenes Thema für sich. Grundsätzlich spricht nichts gegen ihren Einsatz bei mobilen Geräten, da diese Visualisierungen sich für viele Anwendungsgebiete eignen. Die meisten Visualisierungen müssen wegen der kleinen Bildschirme und unterschiedliche Auflösungen bei mobilen Geräten angepasst werden. Besonders die Visualisierungen aus dem Bereich Statistik können mit wenig Aufwand im mobilen Bereich einsatzfähig sein. Dies bekräftigt das Faktum, dass diese Visualisierungen von den meisten Android-Visualisierungsbibliotheken angeboten werden. In Tabelle 2 sind die Visualisierungen dargestellt, die in diesem Kapitel beschrieben werden.

3.5.2.1 Liniendiagramm

Mit einem Liniendiagramm wird grafisch in Form einer Linie der funktionelle Zusammenhang zweier Merkmale dargestellt. Es gibt unterschiedliche Arten zur Darstellung von Liniendiagrammen. Eine Art ist die Verknüpfung der Punkte durch gerade Linien. Eine andere Art ist der Ausgleich der Linie anhand einer mathematischen Funkion. Je mehr Punkte bei dieser Methode für ein Liniendiagramm vorhanden sind, desto genauer ist der Linienverlauf. In Abbildung 12 ist ein Beispiel für eine Kurvendarstellung von einigen gemessenen Punkten gegeben.

Implementierung in vorhandene Android Bibliotheken: AchartEngine, Androidplot, GraphView, Holo-GraphLibrary, charts4j.

Implementierung in prefuse: ja, es wurde im unterschiedliche Projekte implementiert. In VisuExplore[13], und TimeBench[14]

---

13 Liniendiagramm Implementierung in VisuExplore, abgerufen am 19.11.2015

http://ieg.ifs.tuwien.ac.at/projects/VisuExplore/

14 Liniendiagramm Implementierung in TimeBench, abgerufen am 19.11.2015

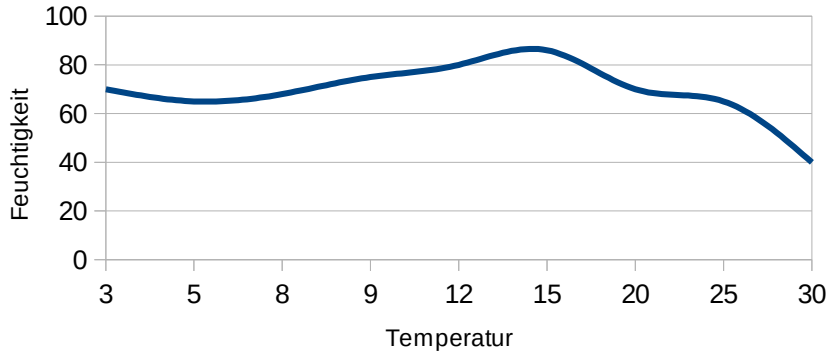

Abbildung 12: Liniendiagramm dargestellt mit einer Kurve

Visualisierung	Implementiert in prefuse	Android-Visualisierungsbibliotheken
Liniendiagramm	ja	AchartEngine, Androidplot, GraphView, Holo-GraphLibrary, charts4j
Säulendiagramm	ja	AchartEngine, Androidplot, GraphView, Holo-GraphLibrary, charts4j
Kreisdiagramm	keine Implementierung bekannt	AchartEngine, Androidplot, Holo-GraphLibrary, charts4j
Streudiagramm	ja	AchartEngine, Androidplot, GraphView, charts4j
Flächendiagramm	ja	AchartEngine, Androidplot, GraphView, charts4j
Diagramme für zeit-orientierte Daten	ja	nein
Fisheye-Diagramm	ja	nein
Tree Map	ja	nein
Graphen	ja	nein
Kartendarstellungen	ja	charts4j
Gantt-Diagramm	ja	nein

Tabelle 2: Vorkommen der Visualisierungen in prefuse und bei Android-Visualisierungsbibliotheken

14 https://github.com/ieg-vienna/TimeBench/blob/master/demo/timeBench/demo/vis/LinePlotDemo.java

3.5.2.2 Säulendiagramm

Das Säulendiagramm, bei sehr schmalen Säulen auch Stabdiagramm genannt, ist ein Diagramm, das durch auf der x-Achse senkrecht stehende, nicht aneinander grenzende Säulen (Rechtecke mit bedeutungsloser Breite) die Häufigkeitsverteilung einer diskreten (Zufalls-)Variable veranschaulicht [35]. In Abbildung 13 ist ein einfaches Säulendiagramm dargestellt.

Implementierung in vorhandene Android-Bibliotheken: AchartEngine, Androidplot, GraphView, Holo-GraphLibrary, charts4j.

Implementierung in prefuse: Ja[15], es wurde im Zuge eines Projektes vom IEG-Institut auf der TU Wien angewendet.

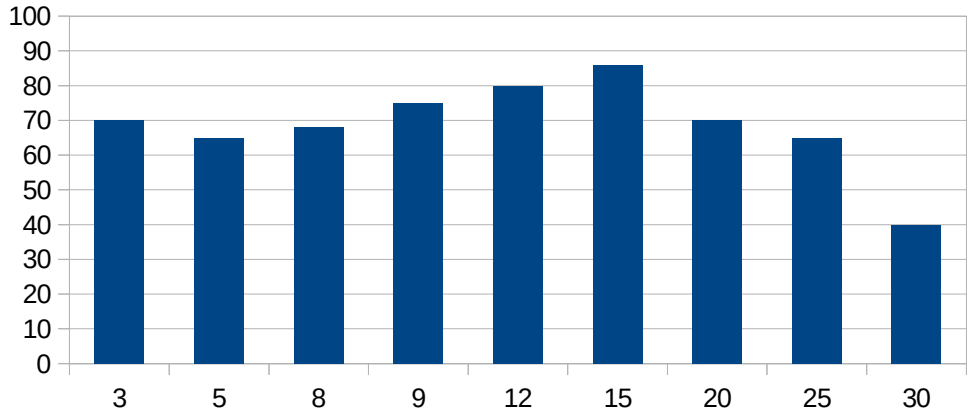

Abbildung 13: Säulendiagramm

3.5.2.3 Kreisdiagramm

Bei einem Kreisdiagramm repräsentiert ein Kreis ein Ganzes (100%) von etwas. Mittels Kreissektoren werden dann die Teile im Kreis (manchmal auch farblich) dargestellt. Dieser Diagramm wird auch Kuchen- oder Tortendiagramm genannt, da die Schnitte eines runden Kuchens dem Kreisdiagramm ähneln. In Abbildung 14[16] ist ein Kreisdiagramm, wo die Verteilung der verschiedenen Android-Versionen angezeigt wird. Die Daten wurden in einer Periode von 7 Tagen bis 02.11.2015 gesammelt.

Implementierung in vorhandene Android-Bibliotheken: AchartEngine, Androidplot, Holo-GraphLibrary, charts4j.

Implementierung in prefuse: keine Implementierung bekannt.[17]

---

15 BarChartRenderer, abgerufen am 27.09.2015
http://ieg.ifs.tuwien.ac.at/projects/trainVis/doc/javadoc/zugvis/BarChartRenderer.html

16 Quelle von Kreisdiagramm Bild, abgerufen am 18.11.2015

http://developer.android.com/about/dashboards/index.html

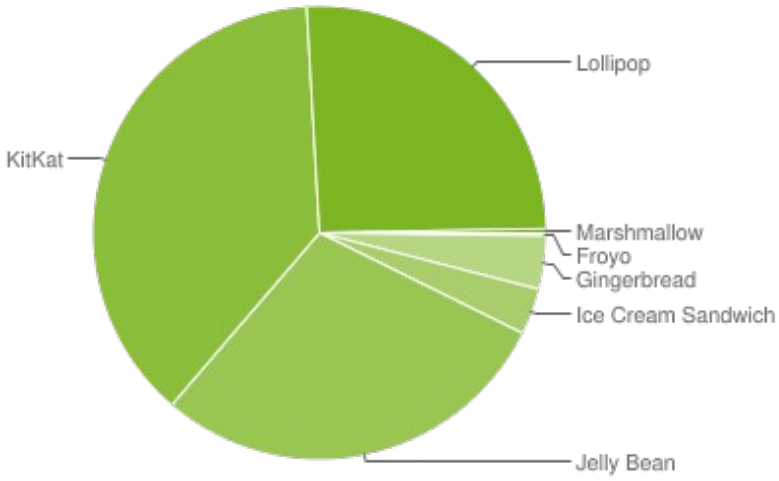

Abbildung 14: Kreisdiagramm – Anteil der verschiedenen Android-Versionen an allen Geräten mit Android OS. Quelle: developer.android.com

3.5.2.4 Streudiagramm

Streudiagramme sind zweidimensionale Grafiken. In ihr werden Wertepaare mit zwei Merkmalen abgebildet. Mit Streudiagrammen wird die Verteilung der Wertepaare beobachtet. Es gibt unterschiedliche statistische Ansätze, um aus der Verteilung und Lage der Paare einen möglichen Zusammenhang zwischen den Merkmalen zu vermitteln. Neben der dargestellten Form (Kreis, Stern, Rechteck, etc.) und Farbe können weitere Merkmale in dem Streudiagramm abgebildet werden. In dem Streudiagramm in Abbildung 15[18] sind Schiffe abgebildet. Die x-Achse steht für die Breite, die y-Achse für die Länge des Schiffes und die Farbe stellt die Art des Schiffes dar.

Implementierung in vorhandene Android-Bibliotheken: AchartEngine, Androidplot, GraphView, charts4j.

Implementierung in prefuse: Ja, wurde von Alexander Rind implementiert[19].

17 Das Sunburst-Diagram http://vialab.science.uoit.ca/portfolio/docuburst-visualizing-document-content-using-language-structure (Abgerufen am 13.12.2015) ist ähnlich aufgebaut, weshalb eine Implementierung von Kreisdiagramm im prefuse voraussichtlich einfach wäre.

18 Quelle vom Streudiagramm Bild, abgerufen am 27.09.2015 https://de.wikipedia.org/wiki/Streudiagramm#/media/File:Lang-breit.svg

19 Streudiagramm, abgerufen am 27.09.2015 http://www.ifs.tuwien.ac.at/~rind/w/doku.php/java/prefuse-scatterplot

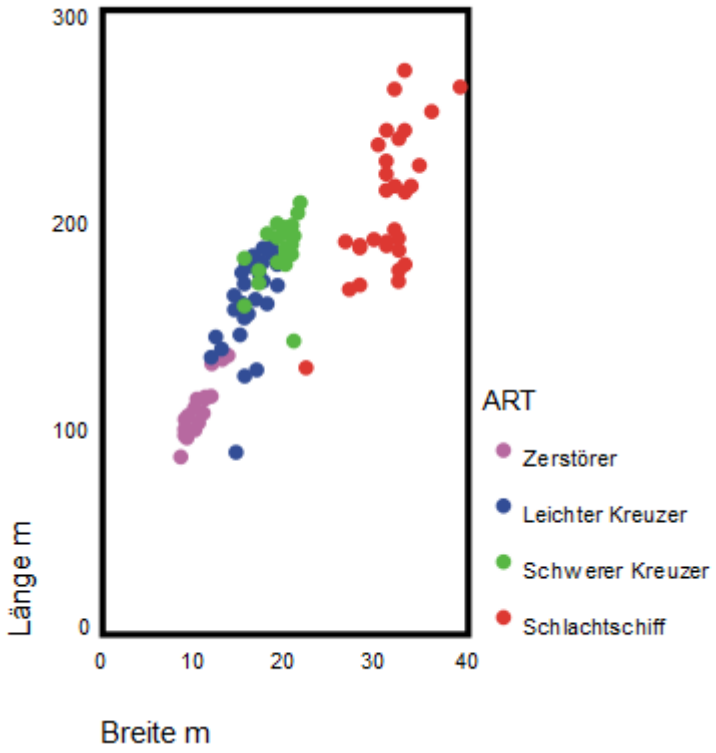

Abbildung 15: Beispiel eines Streudiagramms, in dem die Länge und Breite von verschiedenen Artillerieschiffen dargestellt ist (Quelle: Wikipedia)

3.5.2.5 Flächendiagramm

Das Flächendiagramm ähnelt sehr dem Liniendiagramm (siehe 3.5.2.1). Bei diesem Diagramm werden in Form von farblichen Flächen Mengen dargestellt. In Abbildung 16 ist ein solches Diagramm aus Wikipedia[20] dargestellt.

Implementierung in vorhandene Android Bibliotheken: AchartEngine, Androidplot, GraphView, charts4j.

Implementierung in prefuse: Ja, in prefuse gibt es die Klasse *StackedAreaChart* aus dem Paket *prefuse.action.layout*. Mit dieser Klasse kann ein solches Diagramm erstellt und konfiguriert werden.

20 Überlagertes Flächendiagramm, abgerufen am 27.09.2015 https://de.wikipedia.org/wiki/Flächendiagramm#/media/File:US_and_USSR_nuclear_stockpiles.svg

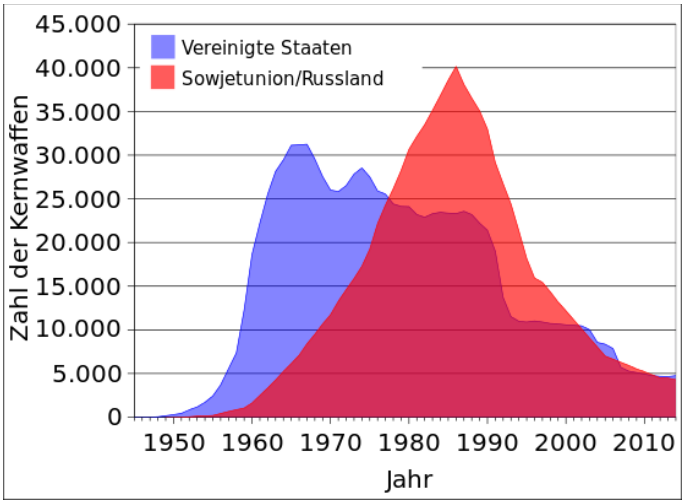

Abbildung 16: Überlagertes Flächendiagramm. Die Anzahl der Kernwaffen, die zwischen 1950 und 2010 in Besitz von Vereinigten Staaten und Sowjetunion/Russland waren. (Quelle: Wikipedia)

3.5.2.6 Diagramme für zeitorientierte Daten

In verschiedenen Gebieten spielt die Zeit eine sehr wichtige Rolle. Momentan wird in der Visualisierungswissenschaft viel zum Thema Zeit als spezielles Merkmal für Visualisierungen geforscht. Daher gibt es verschiedene Ansätze für Visualisierungen in verschiedenen Gebieten. Aigner et al [3] beschreiben die Rolle der Zeit als eigene Dimension in Visualisierungen. Bei dem Buch [4] wurde eine umfassende Klassifizierung dieser Diagramme durchgeführt. Diese Klassifizierung ist auch online[21] ersichtlich. In Abbildung 17 finden sich einige Diagramme, bei denen die Zeit eine große Rolle spielt. Rind et al haben ein spezielles Tool namens TimeBench [32] für Visualisierungen zeitspezifischer Daten entwickelt

Implementierung in vorhandene Android-Bibliotheken: keine Implementierung in den im Kapitel erwähnten Bibliotheken.

Implementierung in prefuse: TimeBench [32] ist eine Erweiterung von prefuse, mit der viele zeitorientiere Diagramme erstellt werden können.

3.5.2.7 Fisheye-Diagramm

> „Fish-eye lenses magnify the center of the field of view, with a continuous fall-off in magnification toward the edges. Degree-of-interest values determine the level of detail to be displayed for each item and are assigned through user interaction.“ [37] - Seite 81

21 The TimeViz Browser, abgerufen am 18.11.2015
http://survey.timeviz.net

Bei solchen Diagrammen werden einige Einträge (meist die, die im Zentrum des Bildes sind) stark vergrößert dargestellt. Die Idee dahinter ist, wichtige Einträgen visuell hervorzuheben. In Abbildung 18 ist ein Fisheye-Menü abgebildet, in dem nur die Einträge vergrößert dargestellt werden, auf welche die Maus fokussiert ist.

Implementierung in vorhandene Android-Bibliotheken: keine Implementierung in den in Kapitel 2 erwähnten Bibliotheken.

Implementierung in prefuse: Ja, es ist bei den prefuse-Beispielen vorhanden: *prefuse.demos.FisheyeMenu*

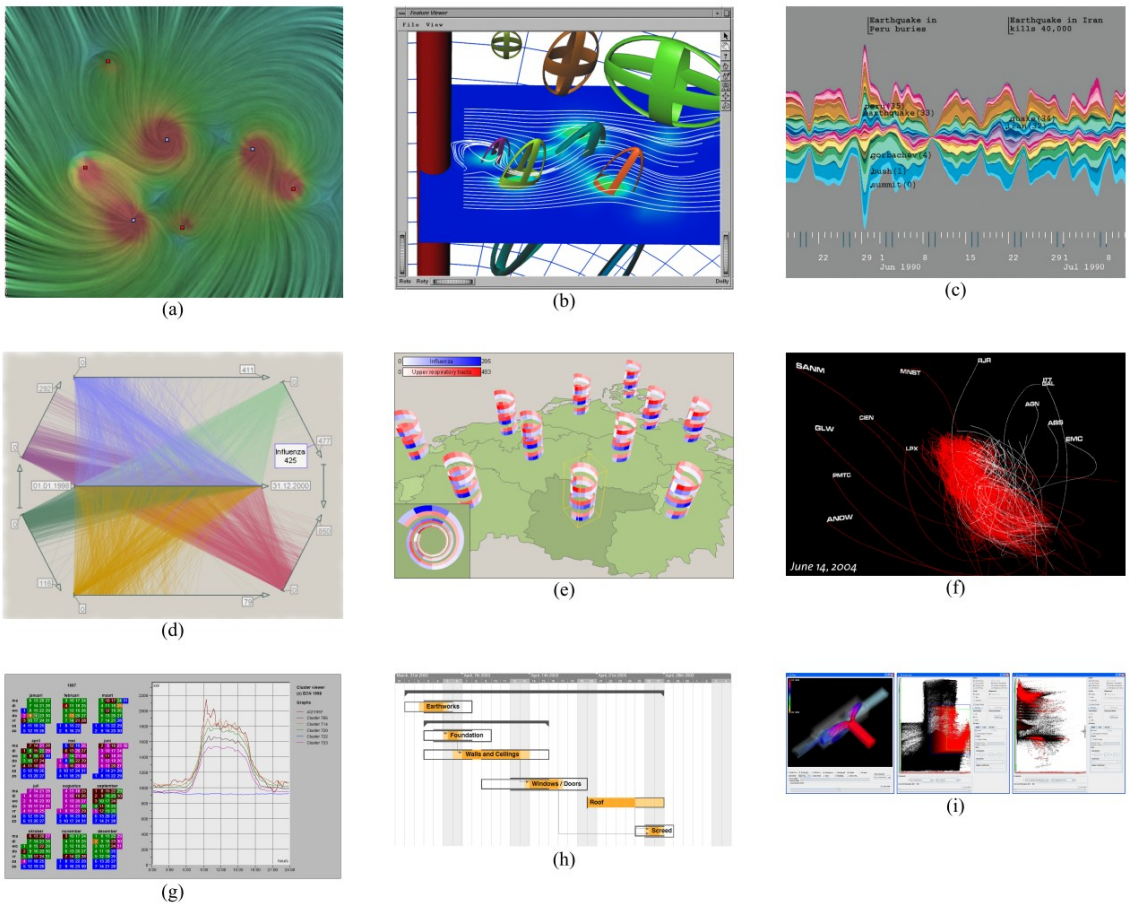

Abbildung 17: Beispiele für Techniken zur Visualisierung zeitorientierter Daten (Quelle: [3])

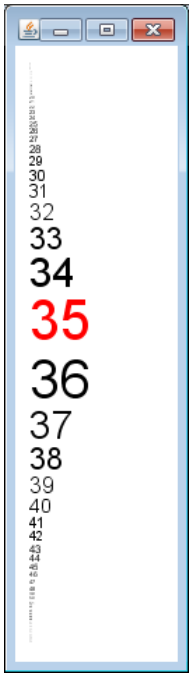

Abbildung 18: Fisheye-Menü (Quelle: prefuse-demo)

3.5.2.8 Tree Map

Tree Map ist eine Visualisierung, bei der Einträge hierarchisch ineinander verschachtelt als Rechtecke dargestellt werden. Es ist sehr gut geeignet, um Verhältnisse zwischen Einträgen in Bezug auf die Größe oder deren Anzahl darzustellen. Tree Maps wurden von Bederson et al.[5] folgendermaßen beschrieben: „Treemaps are a space-filling visualization method capable of representing large hierarchical collections of quantitative data“[33] und „A treemap (Figure 1) works by dividing the display area into a nested sequence of rectangles whose areas correspond to an attribute of the data set, effectively combining aspects of a Venn diagram and a pie chart.“[5], Seite 1

Implementierung in vorhandene Android Bibliotheken: keine Implementierung in den in Kapitel 2 erwähnten Bibliotheken.

Implementierung in prefuse: Ja, es ist bei den prefuse-Beispielen vorhanden: *prefuse.demos.TreeMap*

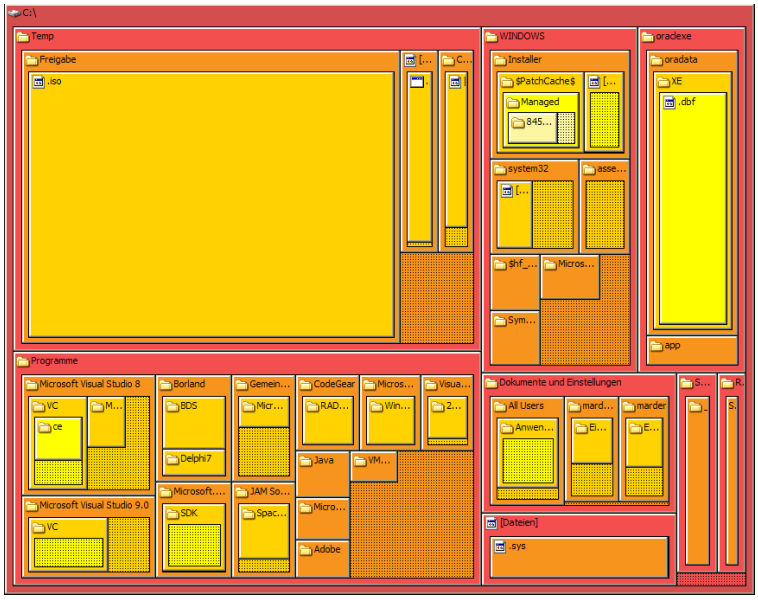

Abbildung 19: Visualisierung eines Verzeichnisbaums mit Hilfe einer Tree Map in TreeSize, einem Programm der Firma JAM Software (Quelle: Wikipedia)

3.5.2.9 Graphen

Ein Graph stellt eine Menge von Objekten, die miteinander in Verbindungen stehen, dar. Die Objekte werden Knoten und die Verbindungen Kanten des Graphs genannt. Graphen werden meistens verwendet, um die Verwandtschaft bzw. die Verbindung zwischen Objekten darzustellen. Ein Beispiel wäre die Zugverbindungen zwischen Städten, bei denen die Städte die Knoten im Graph und die Kanten die Zugverbindungen darstellen. In Abbildung 20 ist eine spezielle Art von Graph, eine Art von Netzdiagramm, dargestellt. Spezielle Graphen sind die sogenannten Bäume: Bei diesen gibt es nur einen Knoten, der keinen Vorgänger hat; dieser wird als Wurzelknoten bezeichnet. Jeder Knoten im Baum hat ein oder mehrere Kinder. In Abbildung 21 ist ein Baumdiagramm dargestellt.

Implementierung in vorhandene Android-Bibliotheken: keine Implementierung in den in Kapitel 2 erwähnten Bibliotheken.

Implementierung in prefuse: Ja, es ist in den prefuse-Beispielen vorhanden. Paket: *„prefuse.demos“ ,* Klassen: GraphView, RadialGraphView und TreeView

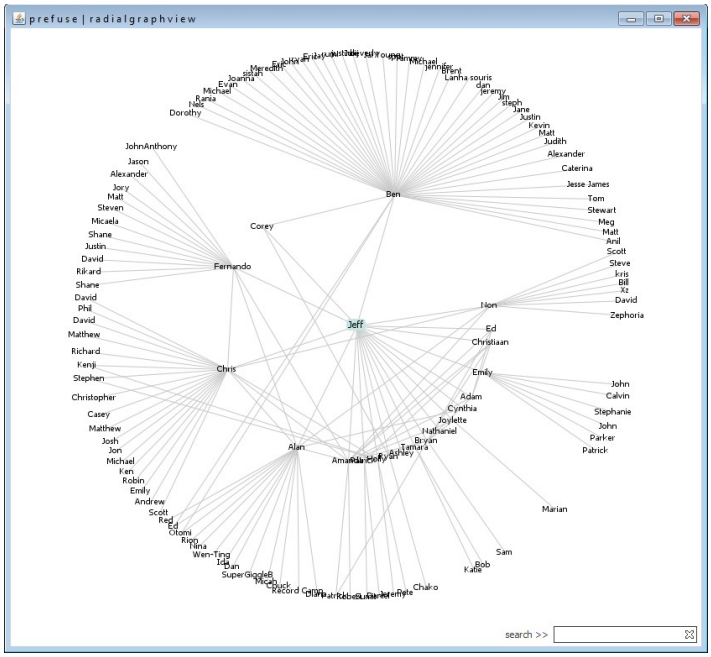

Abbildung 20: Graphen, RadialGraphView (Quelle: prefuse-Demos)

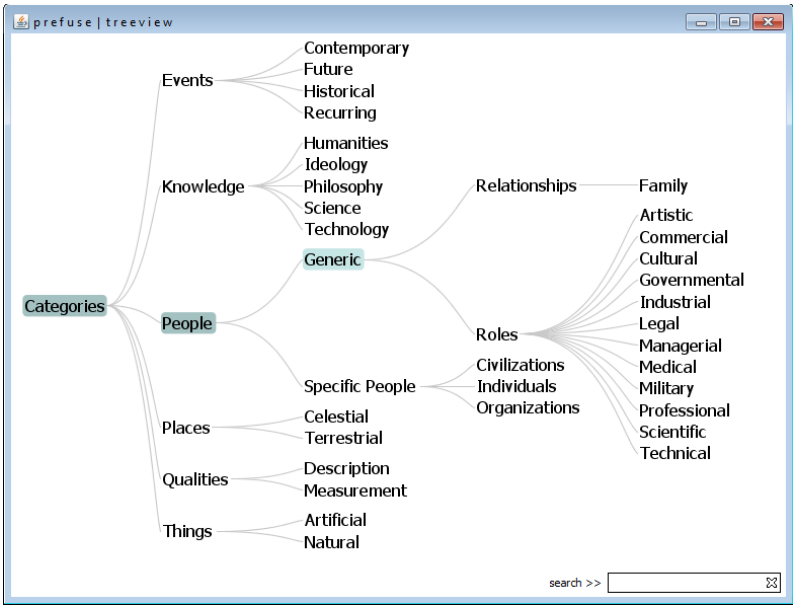

Abbildung 21: Graphen, Baumdiagramm (Quelle: prefuse-Demos)

3.5.2.10 Kartendarstellungen

Bei dieser Art der Visualisierung werden geografische Karten benutzt, um bestimmte raumbezogene Merkmale darzustellen. Abbildung 22 zeigt eine Kartendarstellung der Postleitzahlen in den USA.

Implementierung in vorhandene Android-Bibliotheken: charts4j.

Implementierung in prefuse: Ja, es ist in den prefuse-Beispiele vorhanden. Paket: „*prefuse.demos“ ,* Klasse: ZipDecode

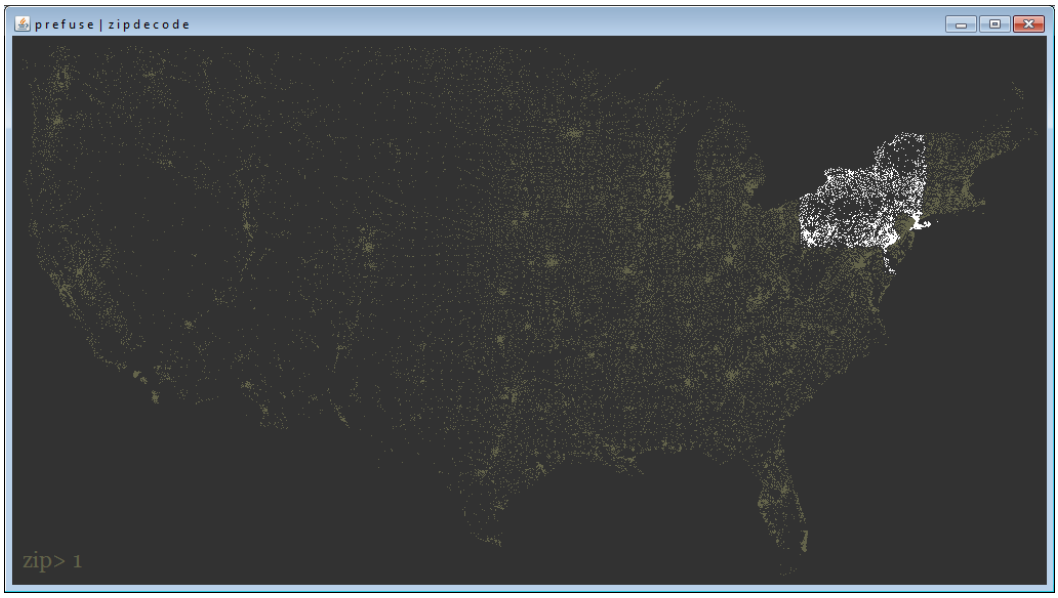

Abbildung 22: Kartendarstellung der Daten - Postleitzahlen in den USA (in Weiß die Postleitzahlen, die mit der Zahl 1 beginnen) (Quelle: prefuse-Demos)

3.5.2.11 Gantt-Diagramm

Das Gantt-Diagramm ist ein Diagramm, das bei der Planung von Projekten aller Art verwendet wird. In diesem Diagramm wird die zeitliche Abfolge von Aktivitäten im Projekt in Form von Balken dargestellt. Die Zeit wird auf der x-Achse abgebildet. Die Aktivitäten werden in der ersten Spalte dargestellt. Die Länge der Balken stellt die Dauer der Aktivität dar, die Höhe der Balken hat keine Bedeutung. Die Abhängigkeit zwischen den Aktivitäten wird mit Pfeilen visualisiert. In Abbildung 23[22] ist ein einfaches Gantt-Diagramm dargestellt.

Implementierung in vorhandene Android-Bibliotheken: keine Implementierung in den in Kapitel 2 erwähnten Bibliotheken.

Implementierung in prefuse: Im Zuge einer Bakkalaureatsarbeit [38] wurde eine Art von Gantt-Diagramm implementiert. Zusätzlich gibt es bei Timebench [32] eine Implementierung[23].

---

22 Gantt-Diagramm, abgerufen am 05.10.2015
https://de.wikipedia.org/wiki/Gantt-Diagramm#/media/File:Gantt_diagramm.svg

23 Gantt-Diagramm in Timebench, abgerufen am 05.10.2015
https://github.com/ieg-vienna/TimeBench/blob/master/demo/timeBench/demo/vis/GanttDemo.java

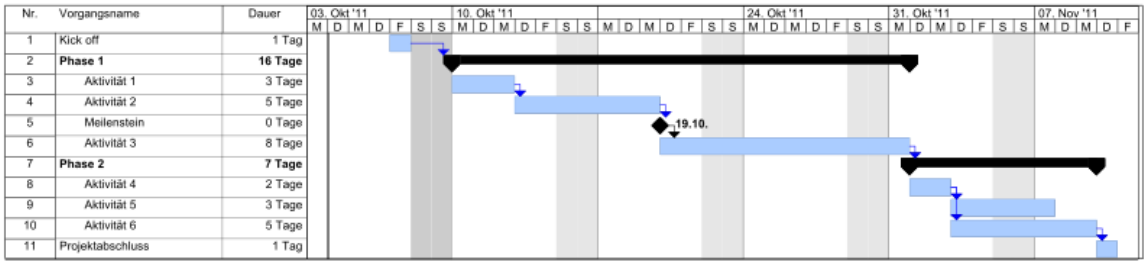

Abbildung 23: Gantt-Diagramm (Quelle: Wikipedia)

3.5.3 Informationsvisualisierungstools

Da die Visualisierungswissenschaft keine neue Wissenschaft und nicht unbedingt mit dem Computer verbunden ist, ist auch die Palette von dazugehörigen Tools relativ umfangreich und auch relativ alt. Das einfachste nicht computerunterstützte Tool sind Papier und Stift. Abbildung 24 zeigt eine Grafik, die 1812 mit Papier und Stift von Joseph Minard erzeugt wurde.

Zu den meist verbreiteten und bekanntesten computerunterstützten Tools zählt Mircrosoft **Excel**. Dabei handelt es sich um ein Tabellenkalkulationsprogramm, das unter anderem eine Vielzahl von Diagrammen wie Linien-, Balken-, Punkt-, Torten-, Streu-, Netzdiagramme und viele andere zur Verfügung stellt.

The InfoVis Toolkit [16] ist ein Toolkit ähnlich wie prefuse. Es bietet viele und meist bekannte Visualisierungen wie Streudiagramme, Zeitreihen (time series), parallele Koordinaten, Tree Maps, Graphen, etc. In Abbildung 25 sind einige durch dieses Toolkit erstellte Beispielsvisualisierungen präsentiert.

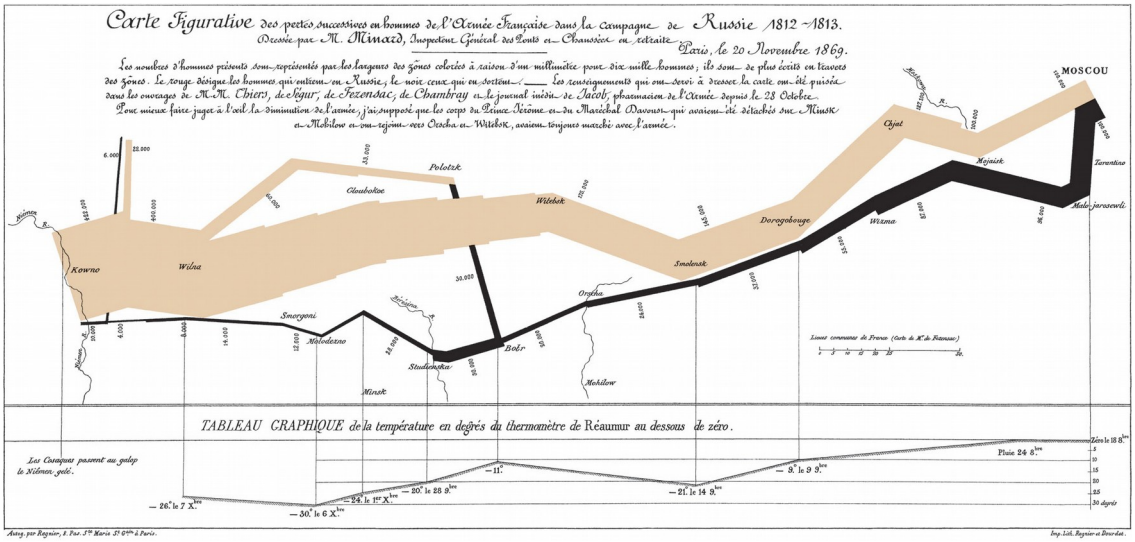

Abbildung 24: Minards Grafik über Napoleons Russlandfeldzug 1812 (Quelle: Wikipedia)

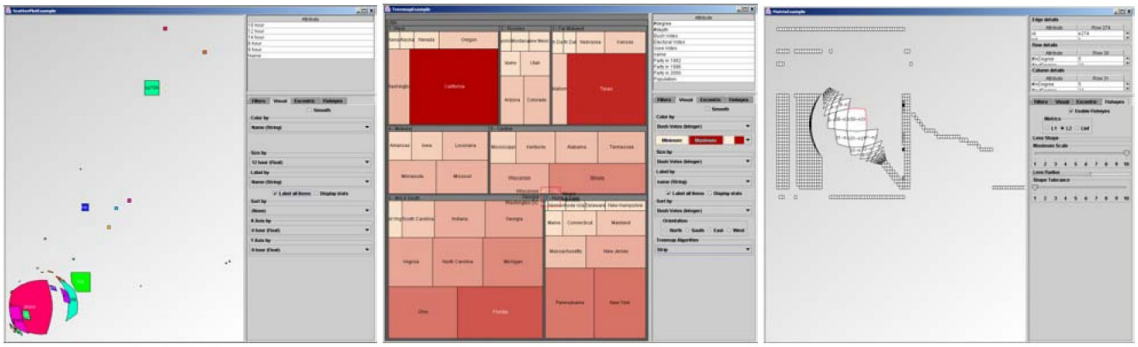

Abbildung 25: Beispiel von Visualisierungen erstellt durch "The InfoVis Toolkit", Streudiagramm, Tree Map, Graphvisualisierung (Quelle: [16])

InstantAtlas[24] ist ein Tool für das Erstellen von Visualisierungen, die mit geografischen Karten zu tun haben.

FusionCharts[25] ist ein JavaScript-Tool und somit ein Tool für Webseiten im Internet. Es bietet über 90 Diagramme und über 900 Karten an.

Google Chart Tool[26] ist ein Online-Tool von Google für das Erstellen von Diagrammen. Es verfügt über eine API, mit der man Diagramme auf der eigenen Webseite erstellen und anzeigen kann. Die Berechnung und Erstellung der Diagramme geschieht auf den Google-Servern, d.h. für die Erstellung der Diagramme ist eine Internetverbindung notwendig. Google liefert am Ende eine PNG-Datei, die man auf der Webseite einbetten kann. Die Liste der angebotenen Diagramme ist mit ca. 30 Diagrammen relativ umfangreich. Unter anderem werden Säulendiagramme, Streudiagramme, Liniendiagramme, Flächendiagramme, Netzdiagramme, Kreisdiagramme, Kartendarstellung, Tree Map, Gantt-Diagramme, Zeitlinien, usw. angeboten. Mittels JavaScript können bei den Diagrammen Benutzerinteraktionen eingebaut werden.

R[27] ist ein Tool für statistische Berechnungen und statistische Grafiken.

Datawrapper[28] ist ein Online-Tool für das Erstellen von interaktiven Diagrammen.

---

24 InstantAtlas, abgerufen am 10.08.2015

http://www.instantatlas.com/

25 FusionCharts, abgerufen am 10.08.2015

http://www.fusioncharts.com/

26 Google Charts, abgerufen am 10.08.2015
https://developers.google.com/chart/interactive/docs/

27 R, abgerufen am 10.08.2015 - https://www.r-project.org/

28 Datawrapper, abgerufen am 10.08.2015 - https://datawrapper.de/

Tableau[29] ist ein kommerzielles Visualisierungstool. Es ist eine Geschäftsanalyse-Anwendung und es bietet verschiedene Visualisierungen mit verschiedenen Einstellungen an.

Leaflet[30] ist eine Open Source JavaScript-Bibliothek für die Erstellung von interaktiven Karten. Daher eignet es sich für den Online-Einsatz. Es ist eine kleine, aber einfache, performante und benutzerfreundliche Bibliothek.

iCharts[31] ist eine cloudbasierte Visual Analytics-Plattform, die eine schnelle Visualisierung von komplexen Geschäftsinformationen, dynamische Datensuche und vieles mehr ermöglicht.

3.5.4 Software-Design-Patterns für Informationsvisualisierung (Heer und Agrawala Paper)

Die relevanten Software-Design-Patterns für Informationsvisualisierung haben Heer und Agrawala [21] erläutert. All diese Patterns und Basis-Design-Patterns stehen in enger Verbindung zueinander. Heer und Agrawala [21]stellen diese Verbindung durch ein übersichtliches Diagramm dar (siehe Abbildung 26).

29 Tableau, abgerufen am 10.08.2015 - https://public.tableau.com/s/

30 Leaflet, abgerufen am 18.11.2015 - http://leafletjs.com/

31 iCharts, abgerufen am 18.11.2015 - http://icharts.net/

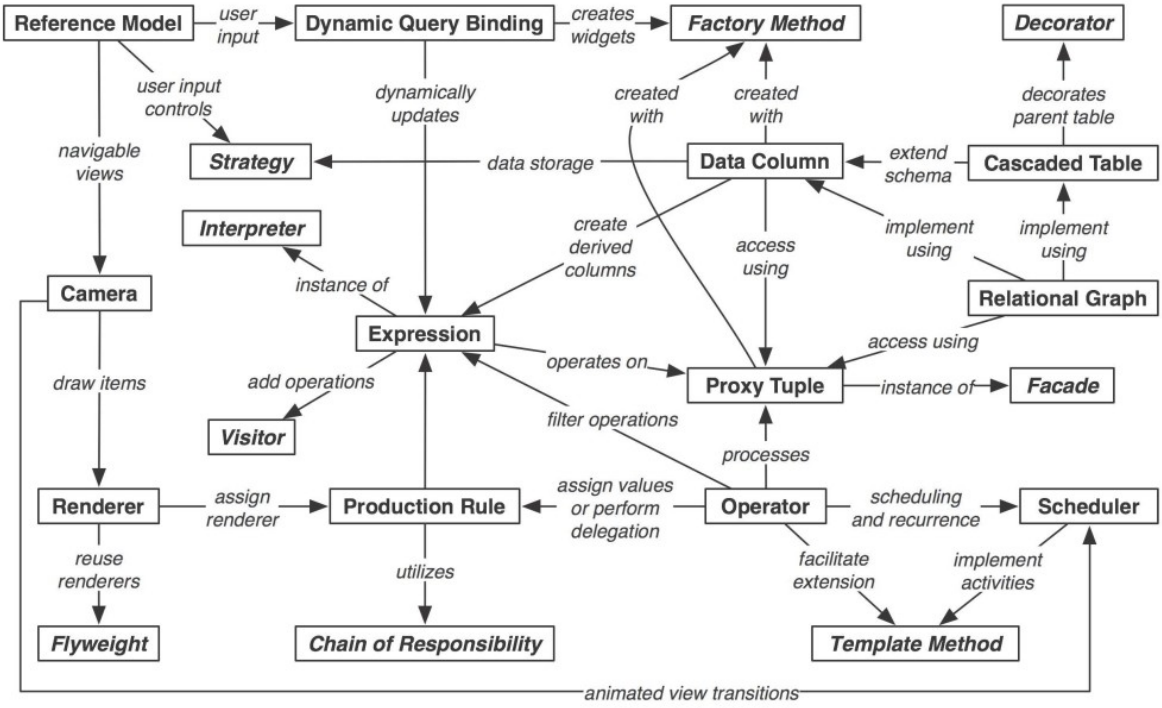

Abbildung 26: Der Graph bildet den Zusammenhang zwischen den Design-Patterns ab, um einen Überblick zu geben, wie die verschiedenen Patterns auf andere angewendet werden können bzw. wie die verschiedenen Patterns einander ergänzen. Patterns in Kursivschrift (z.B. Flyweight) stammen von Gamma et al. [17] Um das Diagramm zu vereinfachen, wurden die Patterns, die häufig von den Visualisierungs-Patterns verwendet worden sind (wie z.B. Observer), weggelassen [21]

3.6 prefuse

3.6.1 Kurze Beschreibung

prefuse [22] ist ein Visualisierungs-Framework für das Erstellen von zweidimensionalen Grafiken. Das Framework wurde konzipiert, um Visualisierungen von komplexen und großen Datenmengen mit relativ kleinem Aufwand zu erstellen. Im wissenschaftlichen Bereich der Informationsvisualisierung wurde prefuse viel und erfolgreich verwendet. Dies belegt die große Anzahl von wissenschaftlichen Publikationen, die prefuse entweder zitieren oder direkt für ihre Forschungszwecke verwenden. Im Folgenden nur eine kurze Liste dieser Publikationen: [1], [14], [23], [25], [32].

In Abbildung 29 werden verschiedene Anwendungen abgebildet, die mittels TimeBench [32], das wiederum auf prefuse basiert, erstellt wurden.

prefuse ist in Java implementiert und benutzt die Standard-Java-Grafikbibliotheken (wie z.B. java.awt.Graphics2D). Um die Funktionen von prefuse zu demonstrieren, haben

seine Autor/innen eine Programmbibliothek von Layout-Algorithmen, Navigations- und Interaktionstechniken, integrierter Suche und vieles andere mit prefuse mitgeliefert. Zusätzlich zu Demonstrationszwecken beinhaltet prefuse selbst einige fertige Grafiken für Streudiagramme, Graphen, Bäume, Timelines, usw. In den Abbildungen 28 und 27 sind zwei Bespiele, die zusammen mit prefuse mitgeliefert werden, dargestellt. Diese bieten auch Benutzerinteraktionen an, wie z.B. beim GraphView (siehe Abbildung 28), bei dem ein Knoten durch Ziehen neu positioniert werden kann. Dadurch werden auch die übrigen Knoten automatisch neu positioniert. Beim TreeView (siehe Abbildung 21) werden beim Klick auf einen Knoten die unteren Knoten auf- bzw. zugeklappt.

Die Struktur von prefuse ist sehr modular aufgebaut, damit es leicht erweitert und angepasst werden kann. Als Hauptdesign-Pattern benutzt prefuse das Architektur-Design-Pattern „Information Visualization Reference Model“ [21].

In Abbildung 30 haben Giereth und Ertl [18] das Klassendiagramm von prefuse dargestellt, wo indirekt die verwendeten Design-Patterns sichtbar sind.

Abbildung 27: Prefuse Demo – DataMountain

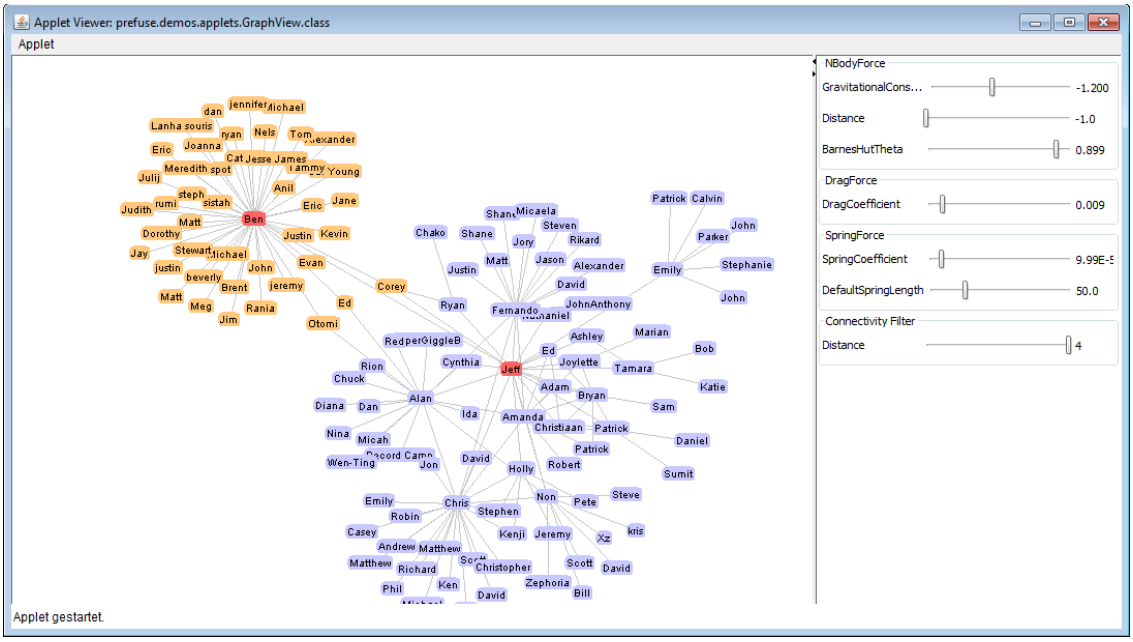

Abbildung 28: Prefuse Demo – GraphView

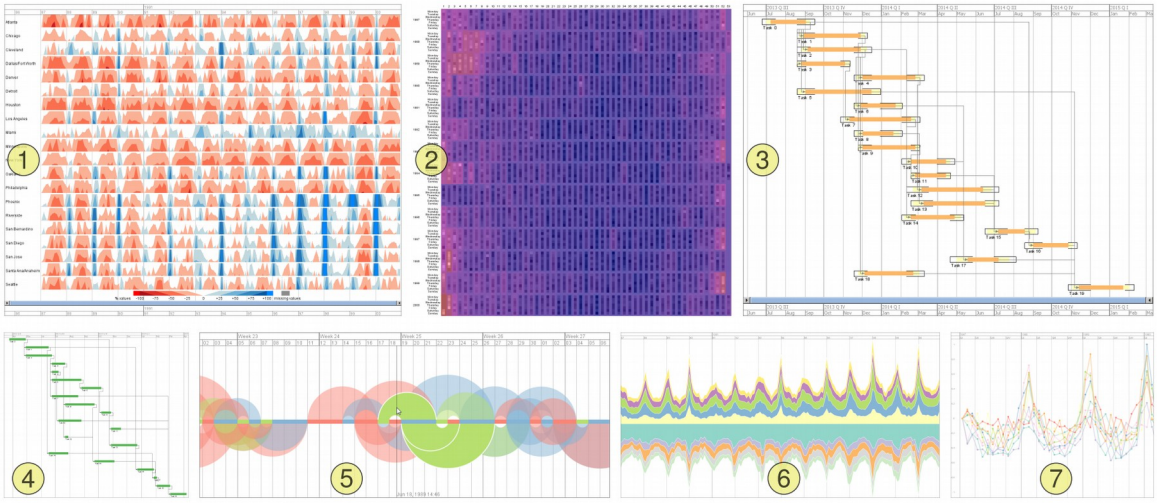

Abbildung 29: Beispielapplikationen erstellt mittels TimeBench: (1) Monatliche Gesundheitsdaten von 20 Städten über 14 Jahre in einem Horizon Graph; (2) Tägliche Gesundheitsdaten über 14 Jahre in einer GROOVE Visualisierung; (3) Projektplan, der "PlanningLines metaphor" benutzt (4) Projektplan in einem Gantt Diagramm (5) ein Arc Diagramm (6) Darstellung der Relationen zwischen 3 Kategorien; (7) Mehrfachliniendiagramm mit Indexierung [32]

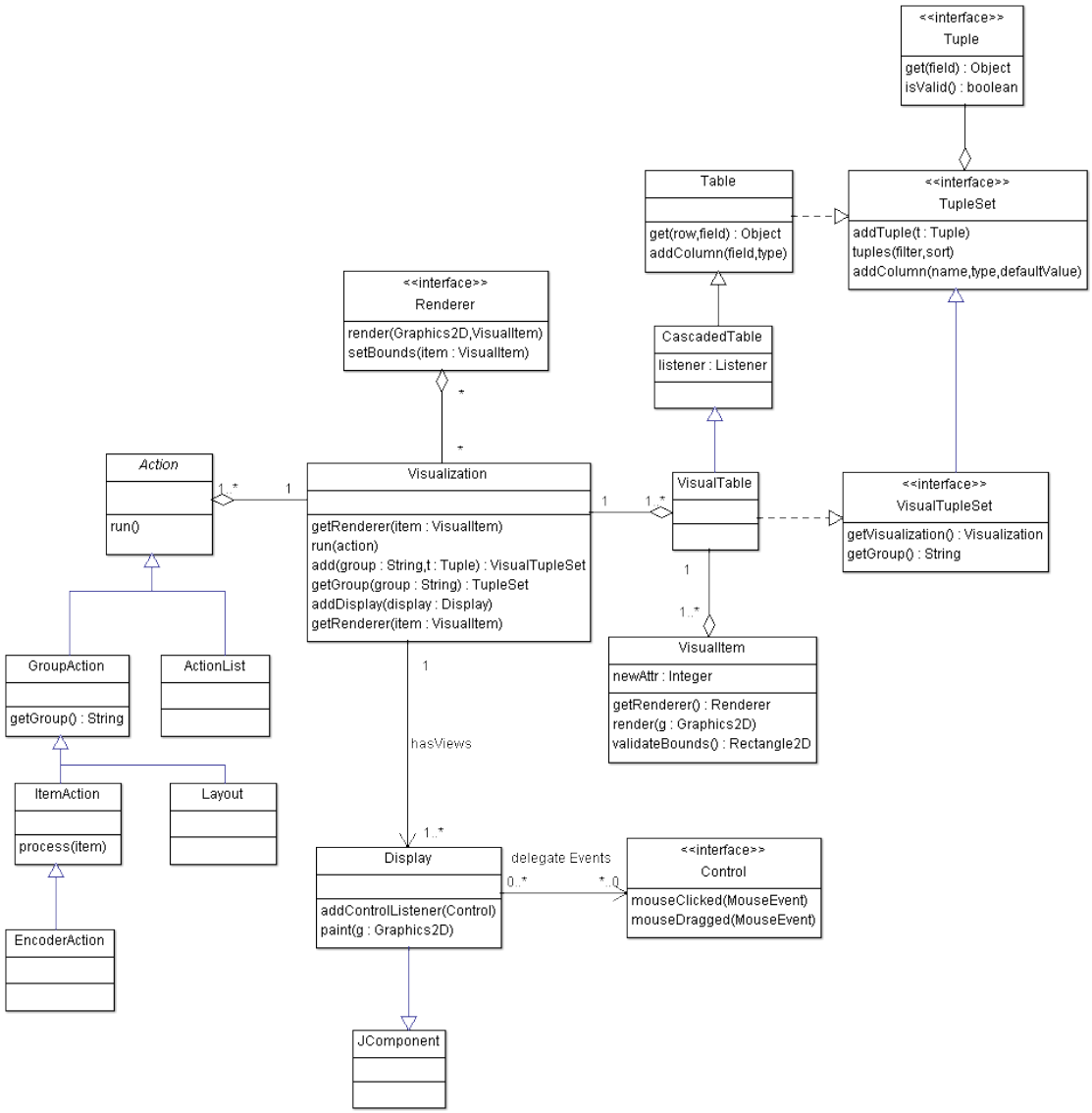

Abbildung 30: Klassendiagramm über die Struktur des prefuse-Frameworks (Basierend auf [18])

3.6.2 Architektur

prefuse implementiert das Architektur-Design-Pattern „Information Visualization Reference Model“ (Abbildung 31). In Abbildung 32 ist eine grobe Struktur der Objekte von prefuse abgebildet. Abbildung 30 bietet ein vereinfachtes Klassendiagramm von prefuse. Die Klasse Visualization ist die Kernklasse von prefuse. Sie beinhaltet und verwaltet die gesamte Visualisierung. Alle Transformierungen wie Filtern und Rendern werden von ihr gesteuert. Ferner werden ihr die Instanzen von Display, Renderer, Action und VisualTable zugeordnet. VisualTable-Objekte sind Objekte für die visuelle Strukturierung und Vorbereitung der Daten für das Rendering. Sie erweitern die Daten aus den Table-Objekten für die visuelle Darstellung mit zusätzlichen Spalten wie z.B. Position, Größe, Farbe und

Form. Die Einträge von VisualTable sind Instanzen von VisualItem. Die Transformierungen (wie z.B. Filtern), Visual-Mapping (wie z.B. Zuordnung von Position, Größe, Farbe, Form, etc.) und Grafiktransformierungen (wie z.B. Verschieben der Grafik, Animationen, Zoomen, etc.) werden über sogenannte Actions ausgeführt. Diese werden in einer Container-Klasse ActionList zusammengefasst und über die Klasse Visualization ausgeführt. Die VisualItem-Objekte werden über einen zugeteilten Renderer, der dann die Einträge in einen oder mehrere Displays zeichnet, abgearbeitet. Die Klasse Display repräsentiert das View aus dem Design-Pattern „Information Visualization Reference Model“. Mindestens eine Instanz dieser Klasse wird der Visualization übergeben. Sie ist eine Subklasse von JComponent und kann somit in jede Java Swing-Applikation integriert werden. Die Hauptaufgabe dieser Klasse ist das Verwalten vom Zeichnen der visuellen Einträge. Sie überprüft bei jedem Eintrag, ob er sich in dem visuellen Bereich der Zeichnung befindet und entscheidet anhand dessen, ob der Eintrag überhaupt gezeichnet werden soll oder nicht. Falls der Eintrag innerhalb der Grenzen der Zeichnung liegt, ruft die Klasse Display den entsprechenden Renderer auf. Die Display-Klasse bietet zusätzliche Hauptmechanismen für View-Navigation wie Zoomen, Rotieren, Verschieben und einige einfache Animationen an. Weiters werden die Benutzerinteraktionen im Display über ControlListener verwaltet. Bei einem Event (Mausklick oder Tastendruck) wird der entsprechende EventListener aufgerufen. Dieser unterscheidet in der Regel, ob das Event innerhalb eines Eintrages oder in einem leeren Bereich aufgetreten ist. Somit ist eine Interaktion möglich, wenn man auf einen Eintrag klickt, wie z.B. das Verschieben eines Eintrags, Anzeige von zusätzlichen Informationen, etc.

3.6.2.1 Verwendete Design-Patterns

Information Visualization Reference Model – In den Abbildungen 31, 32 und 33 ist Information Visualization Reference Model aus unterschiedlichen Aspekten der prefuse-Implementierung abgebildet.

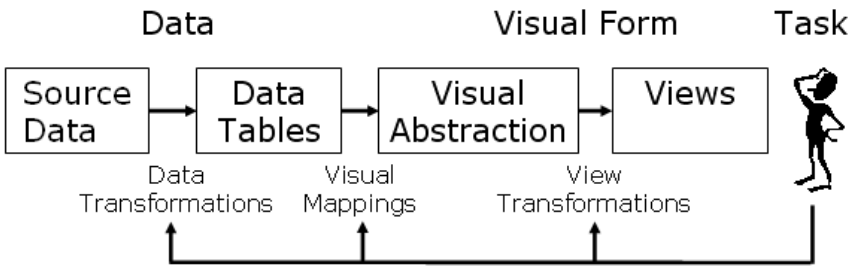

Abbildung 31: Information Visualization Reference Model (Quelle: http://prefuse.org)

Source data: Dies repräsentiert die originalen Quelldaten. Diese Daten können aus unterschiedlichen Quellen stammen, wie z.B. statische Daten, Daten aus einem Sensor

(z.B. Temperatursensor), etc. Sie werden in der Regel entweder in Datenbanken oder in Dateien gespeichert.

Data transformation: Prefuse bietet verschiedene Komponenten zum Lesen von Daten an, wie z.B. *CSVTableReader* zum Lesen von csv-Dateien, *GraphMLReader* zum Lesen von Graph-XML-Dateien, usw. prefuse bietet auch Unterstützung beim Lesen von Daten in SQL-Datenbanken an. Die gelesenen Daten werden dann von prefuse in abstrakte Daten transformiert und in Datentabellen (*Table*-Objekten) gespeichert.

Data tables: Hierbei handelt es sich um prefuse-spezifische Datenstrukturen für Datenmanipulation und Datenspeicherung. Die Daten werden ähnlich wie bei sql-Datenbanken in Tabellen (*Table*-Objekte mit *TupleSets)* gespeichert. prefuse hat spezielle Datenstrukturen auch für Graphen und Bäume. Jeder Eintrag ist als *Tuple* gespeichert. Zusätzlich gibt es *Node* und *Edge,* die dann Knoten und Kanten bei den Graphen und Bäumen repräsentieren. Diese abstrakten Daten beinhalten keine Informationen über die visuelle Darstellung der Einträge. Stattdessen werden diese Informationen separat in einer eigenen Datenstruktur gespeichert.

Visual mappings & Visual abstraction: Im Allgemeinen werden bei „Visual mappings“ die Daten mit zusätzlichen Informationen wie z.B. Position, Größe, Farbe und Form, für die visuelle Darstellung ergänzt. Diese Informationen werden ähnlich wie „Data tables“ im Tabellenformat verwaltet. Die einzelnen Einträge (auch Kanten und Knoten) werden als *VisualItem* gespeichert. Beim „Visual mapping“ werden folgende *Actions* durchgeführt:

- Filter: Bei dieser *Action* werden die abstrakten Daten in entsprechende visuelle Informationen umgewandelt. Hiermit wird für jedes *Tupel* ein *VisualItem* erzeugt.
- Layout: Hierbei wird für jedes *VisualItem* die Position bestimmt. In prefuse stehen schon einige Algorithmen für Graphen und Bäume wie z.B. *ForceDirectedLayout* oder *RadialTreeLayout* zur Verfügung.
- Assignment: Bei dieser *Action* werden den *VisualItems* die visuellen Informationen wie Größe, Farbe, Form, etc. zugewiesen.

Visual transformations: Bei dieser Phase wird jeder einzelne Eintrag durch den entsprechenden *Renderer* bei einem View gerendert. Hierbei wird entschieden, wie die zugewiesenen visuellen Informationen aus „Visual abstraction“ ausgegeben werden. Jedes *VisualItem* bekommt einen oder mehrere *Renderer* durch *RendererFactory* zugewiesen. Zusätzlich dazu, dass die *Renderer* die Einträge in dem View zeichnen, bestimmen die *Renderer*, ob die Einträge überhaupt gerendert (angezeigt) werden sollen.

Views: prefuse benutzt Java Swing als Visualisierungsmedium.

In Abbildung 32 ist die Anordnung der prefuse java-Pakete in Bezug auf Information Visualization Reference Model abgebildet.

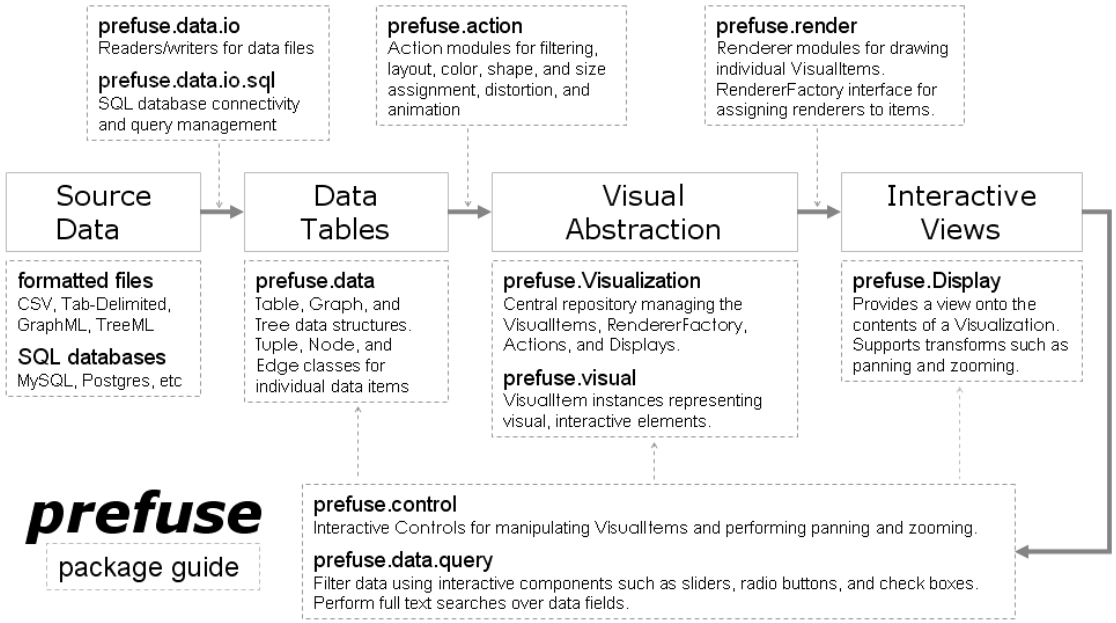

Abbildung 32: Die Anordnung der prefuse java-Pakete in Bezug auf Information Visualization Reference Model (Quelle: http://prefuse.org)

Im Folgenden werden die weiteren Design-Patterns, die in prefuse benutzt werden, erläutert.

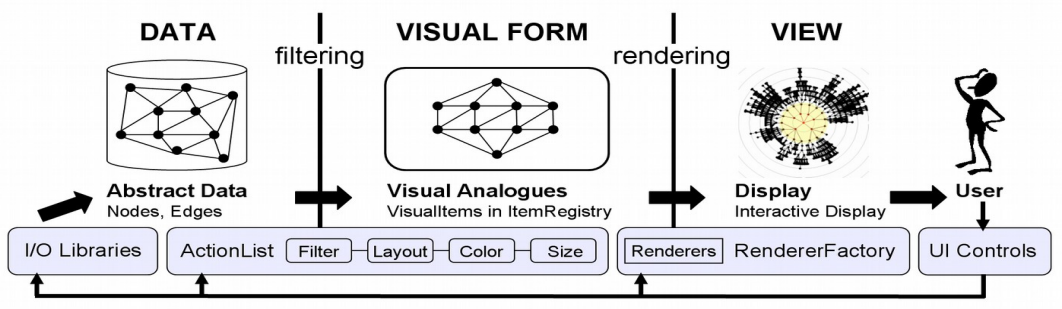

Abbildung 33: Information Visualization Reference Model-Implementierung bei prefuse. Zusammenfassung von unterschiedlichen Actions in der ActionList. Zusammenfassung von unterschiedlichen Renderers in der RendererFactory. Benutzerinteraktion kann Veränderungen zu jeder Stelle im Framework auslösen (Quelle: [22])

Factory Method Pattern [17]: Dieses Pattern wird verwendet, um die Objektinstanzen von unterschiedlichen Klassen dynamisch zu erstellen. Ein Beispiel für die Verwendung dieses Patterns in prefuse ist die Klasse *RendererFactory* (Abbildung 34).

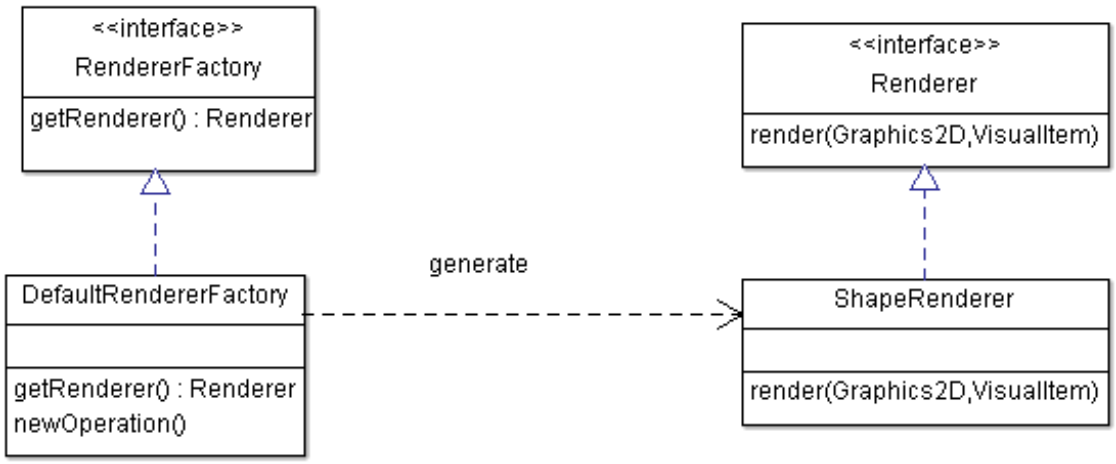

Abbildung 34: Factory Method Pattern in prefuse mit RendererFactory

Strategy Pattern [17]: Dieses Pattern wurde für Controllers im Architektur-Design-Pattern „Information Visualization Reference Model“ [21] verwendet. Mit Strategy Design-Pattern wird das Verhalten oder der Algorithmus einer Klasse dynamisch geändert. Bei prefuse ist bei den Controllers bei gleichen Benutzerinteraktionen unterschiedliches Verhalten der Grafik möglich. In prefuse wird dieses Pattern verwendet, um bei gleichen Benutzerinteraktionen zur Laufzeit dynamisch unterschiedliche Reaktionen zu ermöglichen. In Abbildung 35 ist das Klassendiagramm dargestellt, das die Implementierung dieses Patterns zeigt.

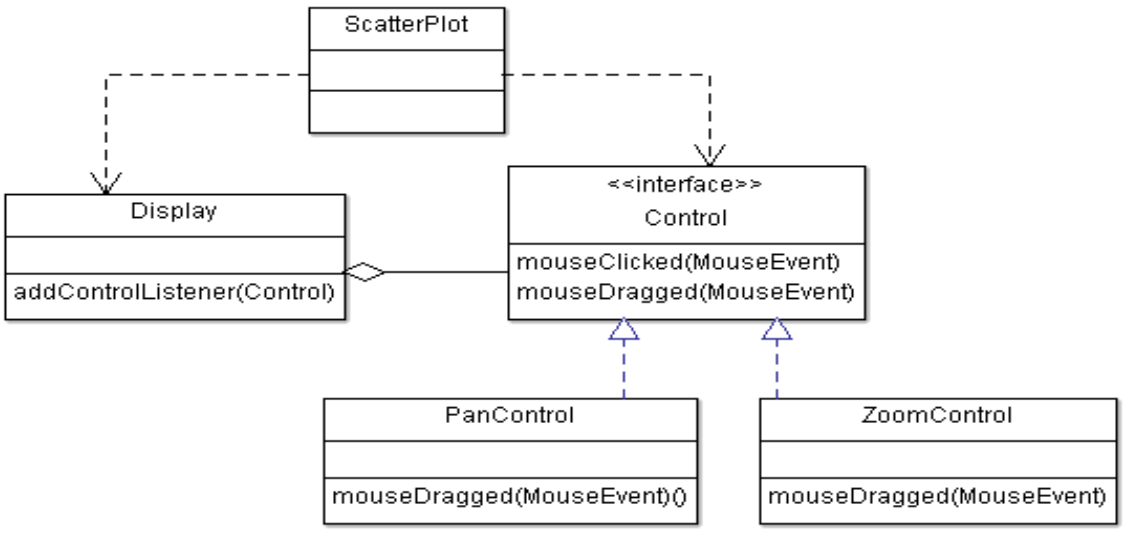

Abbildung 35: Strategy Pattern – Behandlung von Benutzerinteraktion über Controllers

Data Column [21]: Mit diesem Pattern werden relationale Daten zu typisierten Spaltendaten strukturiert. Dies bietet Flexibilität bei Datendarstellung und ermöglicht erweiter-

bare Datenschemas. In Abbildung 36 befindet sich das Klassendiagramm der prefuse-Klassen. Der *Table* besitzt eine oder mehrere Spalten (*Column*). Sie erzeugt die Spalten mittels *ColumnFactory.* Anhand des Observer-Patterns (*ColumnListener*) werden die Änderungen bei den Spalten beobachtet.

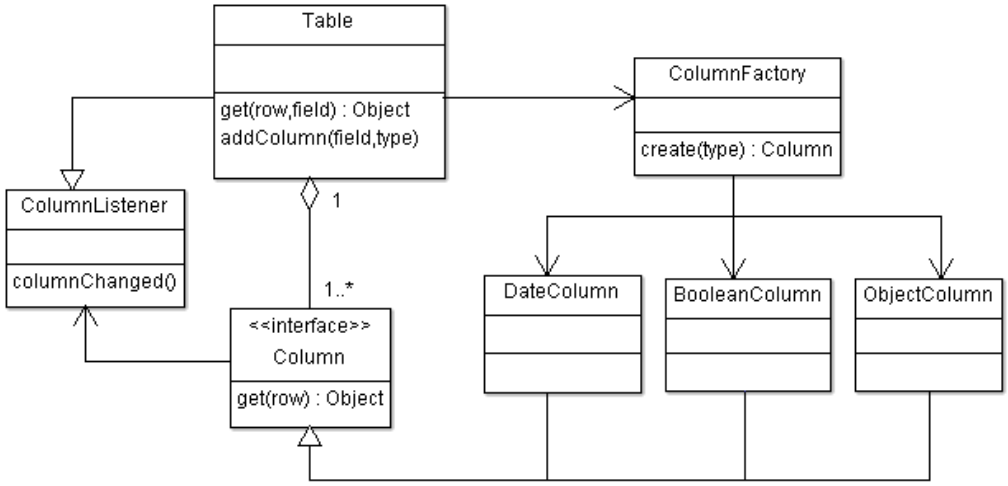

Abbildung 36: Data column pattern in prefuse

Cascaded Tables [21]: Bei diesem Pattern werden die originalen Daten um zusätzliche Metadaten erweitert/ersetzt, ohne die ursprünglichen Daten dabei zu verwerfen. Abbildung 37 zeigt das zugehörige Klassendiagramm dieses Pattern, das in prefuse implementiert ist.

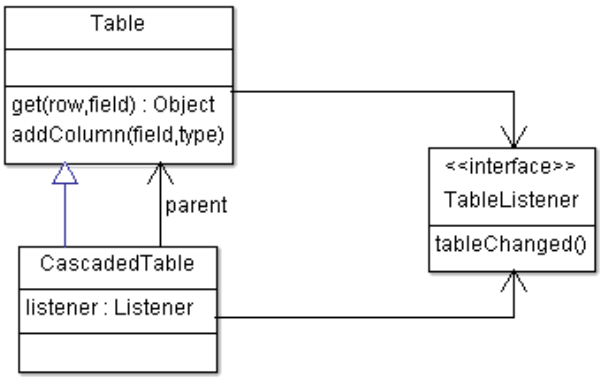

Abbildung 37: Cascaded table design pattern

Relational Graph [21]: Bei diesem Pattern werden die relationalen Daten in Netzwerkstruktur umgewandelt. Dies ermöglicht es, vernetzte Daten performant in Graphen darzustellen. Abbildung 38 zeigt das Klassendiagramm für die Implementierung in prefuse.

Proxy Tuple [21]: Dieses Pattern ist eine Art von Facade Design Pattern[17]. Durch die bereitgestellten Klassen wird die Komplexität des Subsystems vereinfacht, die durch Data Column[21] und Relational Graph[21] entstanden ist. In diesem Pattern (siehe Abbildung 38) wird auf die Daten nicht direkt durch *Table*, *Node*, oder *Edge* sondern durch *Tuple* zugegriffen.

Expression [21]: Das Pattern ist eine direkte Implementierung des Interpreter Patterns [17]. Es stellt eine „Expression Language“ für Datenverarbeitungsaufgaben durch Spezifikation von Abfragen zur Verfügung. In Abbildung 39 ist das Klassendiagramm von Expression Design Patterns abgebildet.

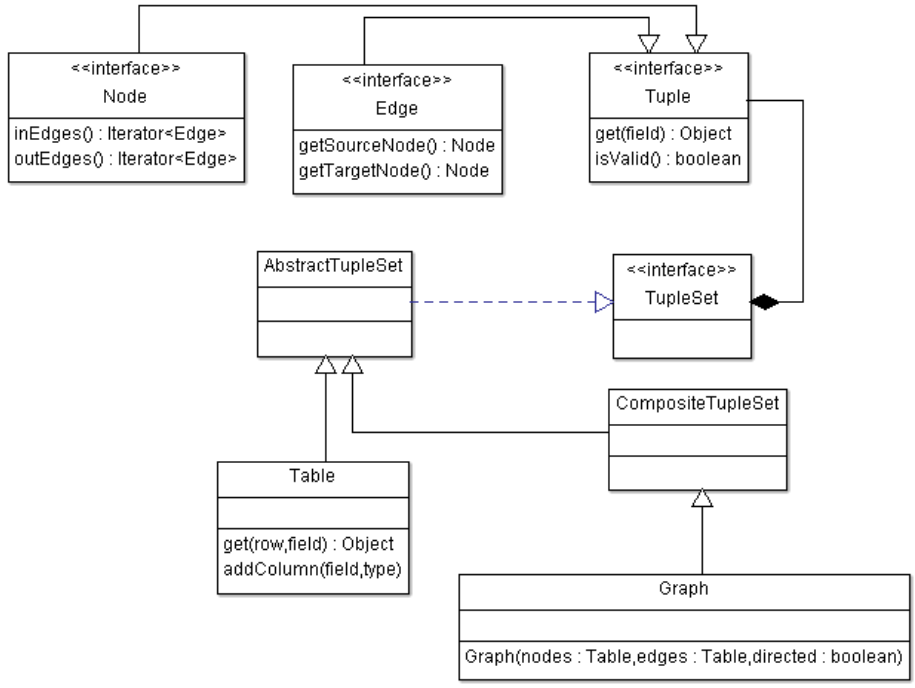

Abbildung 38: Relational Graph und Proxy Tuple Design Pattern

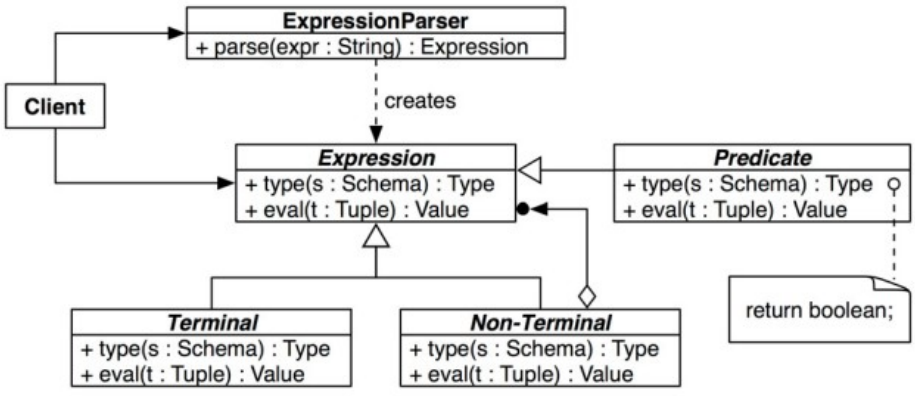

Abbildung 39: Expression Design Pattern (Quelle: [21])

Scheduler [21]: Dieses Pattern bietet die Möglichkeit, Aktivitäten zeitversetzt zu starten bzw. zu wiederholen (Abbildung 40).

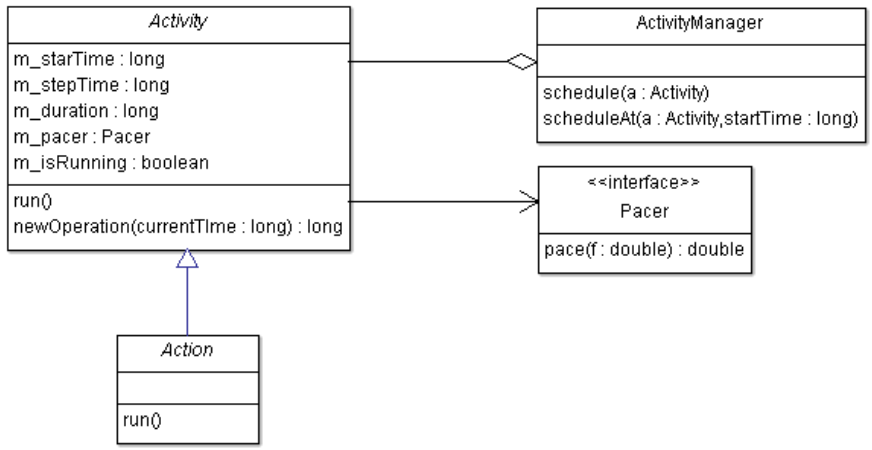

Abbildung 40: Scheduler Design Pattern

Operator [21]: Dieses Pattern ermöglicht, den Datenverarbeitungsprozess in Einzelteile zu zerlegen, um sie dann nach einer bestimmten Reihenfolge auszuführen. Dadurch sind flexible und konfigurierbare Visualisierungen möglich.

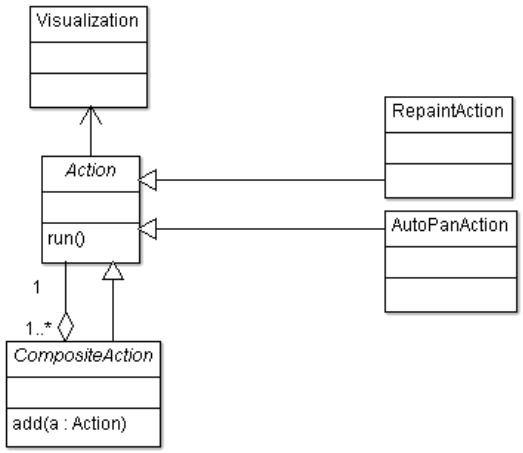

Abbildung 41: Operator Design Pattern

Renderer [21]: Dieses Pattern trennt visuelle Komponenten von ihren Renderings-Methoden. Dies ermöglicht, die visuelle Erscheinung von Daten dynamisch anzupassen. (Abbildung 42)

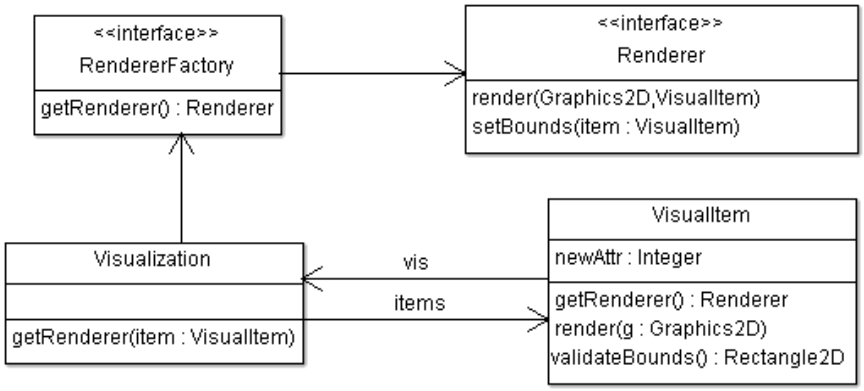

Abbildung 42: Renderer Design Pattern

Camera [21]: Camera bietet die Möglichkeit, dieselben Daten in mehreren Views darzustellen. Dabei zeigt eines der Views den Überblick über alle Daten, während das andere nur einen Teil der Daten abbildet (z.B. nach einem Zoom).

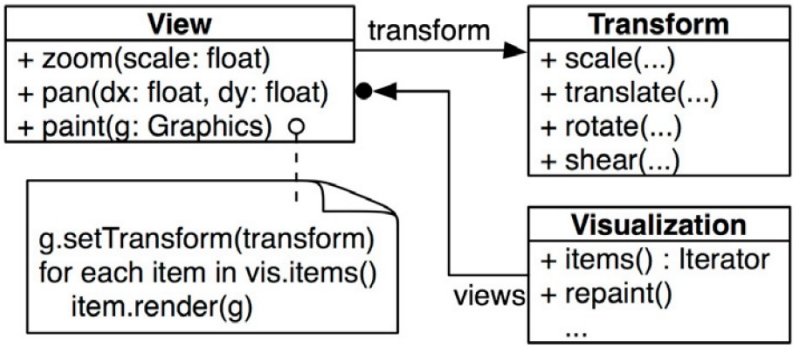

Abbildung 43: Camera Pattern (Quelle: [21])

Durch das breite Spektrum an Anwendungsmöglichkeiten von Design-Patterns, das oben dargelegt wurde, ist die Struktur von prefuse sehr modular aufgebaut. Dies ermöglicht eine leichte Erweiterung des Frameworks. Da die Design-Patterns generell plattformunabhängig aufgebaut sind, können diese durchaus genauso, wie sie in prefuse implementiert sind, in AndroidPrefuse übernommen werden. Dadurch wird erwartet, dass die Portierung auf Android ohne große Komplikationen verläuft.

3.6.3 Schwachpunkte und Verbesserungsvorschläge von bzw. für prefuse

Verwendung von allgemeinen Datentypen: An einigen Stellen kommen in prefuse allgemeine Datentypen (wie z.B. String oder Object) anstatt von spezielleren (eventuell eigenen) Datentypen zum Einsatz. Ein Beispiel ist die Gruppierung bei den *Actions* und Daten. Die Gruppierung wird mit Strings durchgeführt. Dabei wird der Name der Gruppe als String übergeben und dieses wird später in verschiedenen Ebenen mit dem String

angesprochen. Die Schwierigkeit hierbei ist, dass Fehler bei der Programmierung relativ schnell passieren können. Fehler, die aufgrund dessen gemacht werden könnten, sind Tippfehler oder das Übergeben einer String-Variable, die eigentlich nicht für die Gruppierung gedacht ist. Listing 1 zeigt den Fall mit der Gruppierung der *Actions.*

```
VisualTable vt = vis.addTable("data", data);
AxisLayout x_axis = new AxisLayout("data", "NBZ", Constants.X_AXIS,
VisiblePredicate.TRUE);
AxisLabelLayout x_labels = new AxisLabelLayout("xlab", x_axis, boundsLabelsX);

vis.setRendererFactory(new RendererFactory() {
        ...
        public Renderer getRenderer(VisualItem item) {
                return item.isInGroup("ylab") ? arY
                                : item.isInGroup("xlab") ? arX : sr;
        }
});
```

Listing 1: Gruppendefinition und deren Verwendung in prefuse

Anstatt der Verwendung der Strings „data“ und „xlab“ könnte man hier Objekte verwenden, genauer gesagt ein Objekt, das *VisualGroup* heißt. So stellt man sicher, dass man den Methoden nicht irgendein String übergibt, sondern die gewünschte Gruppe.

Keine Generizität: prefuse wurde mit Java 1.4 programmiert, und bei dieser Version war die Unterstützung der Generizität in Java nicht eingebaut. In prefuse werden Listen (z.B. *ArrayList*) und Klassen-Typen (z.B. *Class*) sehr oft verwendet. Hierbei könnten Fehler bei der Programmierung gemacht werden, indem Datentypen gemischt werden, z.B. ein *ArrayList* könnte unterschiedliche Datentypen bekommen, anstatt wie gewünscht nur einen Datentyp.

Polylithisches Design: prefuse wurde bewusst durch viele Design-Patterns implementiert, ebenso beabsichtigt war die Entwicklung in unterschiedlichen Klassen, um eine Erweiterung ohne Umstrukturierung (Refactoring) zu ermöglichen. Die hohe Anzahl an Klassen erhöht allerdings auch die Komplexität, und die Lernkurve ist am Anfang sehr hoch. Dagegen kann man relativ wenig (bis gar nichts) machen, da diese Architektur wie erwähnt bewusst ausgewählt wurde.

3.7 Zusammenfassung

In diesem Kapitel wurde die allgemeine Terminologie, die für den Aufbau dieser Arbeit notwendig ist, kurz erläutert. Die Informationen über Grafikprogrammierung in der Desktop-Umgebung und über Grafikprogrammierung in Android sind Grundlagen für die Portierung von prefuse nach Android. Erst wenn man über ein Grundwissen über die beiden Systeme und die Unterschiede zwischen ihnen verfügt, kann man den Plan für die Portierung erstellen. Aus den obigen Informationen konnte man schließen, dass die unterschiedlichen Kern-Grafikbibliotheken der beiden Systeme eine große Hürde für die

Portierung darstellen. Genauer, die im Android fehlenden AWT- und Swing-Bibliotheken stellen ein Hindernis bei der Portierung dar. Ein wichtiger Faktor dabei ist prefuse selbst.

Außerdem wurde der Aufbau und die Architektur von prefuse genau erklärt. Darin verwendete Design-Patterns wurden im Detail erläutert. Zusammengefasst lässt sich sagen, dass prefuse einige in der Softwareentwicklung allgemeine Design-Patterns verwendet, wie z.B. Factory Method-Pattern und Strategy-Pattern. Aus Sicht der Visualisierung verwendet es eines der bekanntesten Architektur-Design-Patterns in der Visualisierung, „Information Visualization Reference Model“. Bei der Implementierung dieses allgemeinen Design-Patterns haben die Entwickler/innen einige visualisierungsspezifische Design-Patterns wie z.B. Data Column, Cascaded Tables, Expression, etc. verwendet. Bei der Beschreibung dieser Patterns wurde die erste Teilfrage der zweiten Forschungsfrage, „Welche Software Design-Patterns kommen bei (Desktop) prefuse vor?“, beantwortet.

Es gibt Design-Patterns, die abhängig von vielen Faktoren wie z.B. Betriebssystem, Anwendungsgebiet, bestimmte Programmierbibliothken oder ähnliches sind. Die meisten sind aber nicht auf ein Betriebssystem, eine Laufzeitumgebung beschränkt. Einige objektorientierte Programmiersprachen haben spezifische Merkmale, wie z.B. Mehrfachvererbung, die nicht in alle Programmiersprachen vorhanden sind. Es könnte Design Patterns geben, die auf diese spezifische Merkmale aufgebaut sind. Daher können diese Design Patterns nicht eins zu eins in jeder Programmiersprache übernommen werden. Da prefuse und Android dieselbe Programmiersprache verwenden, gibt es zumindest aus dieser Perspektive keine Hindernisse. Das Verwenden der gemeinsamen Programmiersprache ist auch ein Argument dafür, dass die Design-Patterns sich theoretisch in Android anwenden lassen. Die allgemeinen (nicht visualisierungsspezifischen) Design-Patterns wie Factory Method-Pattern und Strategy-Pattern sind bei allen objektorientierten Programmiersprachen und in jeder Umgebung anwendbar. Nach der Analyse der visualisierungsspezifischen Design-Patterns ist ersichtlich geworden, dass keines von ihnen ein umgebungsspezifisches Merkmal verwendet und sie daher in jeder Umgebung, darunter auch in Android, anwendbar sind.

Damit sind die Grundlagen gelegt, um die Forschungsfragen 2 und 3 zu beantworten. Offen bleibt, ob sich die Design-Pattern im mobilen Bereich praktisch anwenden lassen. Dieser Frage soll im nächsten Kapitel nachgegangen werden.

4 AndroidPrefuse

4.1 Beschreibung

AndroidPrefuse ist der Name des Projektes, das nach der Portierung von prefuse zu Android entsteht. Am Ende der Portierung sollte es dieselben Erweiterungsmöglichkeiten wie prefuse aufweisen. Der entwickelte Prototyp sollte als erster Grundstein für die vollständige Portierung dienen. Dieser erste Grundstein ist die Portierung der Architektur von prefuse. Als Nebenziel wurde gesetzt, zu versuchen, so viel wie möglich von prefuse zu übernehmen.

Nach der Portierung (siehe nächstes Kapitel 4.2) ist ein Prototyp entstanden, der die Hauptanforderung (Übernahme von prefuse-Architektur) erfüllt (siehe Kapitel 4.3).

4.2 Durchführung der Portierung

In Kapitel 4.5.1 (ein Unterkapitel von 4.5 „Herausforderungen und deren Lösungen“) wird beschrieben, welcher Ansatz für die Portierung gewählt wurde. Kurz zusammengefasst, wurde entschieden die quelloffenen Klassen aus den Bibliotheken java.swing und java.awt zu übernehmen. Da prefuse nur von diesen zwei Bibliotheken abhängt, sollte die Portierung mit geringfügigen Anpassungen an prefuse umsetzbar sein.

Die Klassen wurden aus zwei Quellen übernommen. Die erste Quelle war der Code der frühen Android Version 2.2.3[32]. Grund für diese Auswahl war erstens die geeignete Lizenz (für mehr siehe Kapitel 4.5.3), zweitens wurden diese Bibliotheken in späteren Versionen von Android entfernt. Da die Veröffentlichung dieser Android-Version bereits einige Jahre her ist (Veröffentlichungsdatum 21.11.2011[33]), stellte sich die Frage, ob der Code zu alt für prefuse ist und als Folge dessen Kompatibilitätsprobleme auftreten würden, also ob prefuse mit einer neueren Java-Version aufgebaut ist und deswegen einige Funktionen in der Java-Version von Android 2.2.3 fehlen könnten.

Android 2.2.3 und ältere Versionen wurden so konstruiert, dass sie die Java-Versionen 5 und 6 unterstützen. Die neuen Versionen ab Android 4.4 unterstützen Teile von Java-Version 7. Also wäre es nur problematisch, wenn prefuse Konzepte Java 7 benutzt. Es konnten keine offiziellen Informationen darüber gefunden werden, unter welcher Java-Version prefuse gebaut worden ist. Die letzte Version von prefuse wurde am 21.10.2007[34] veröffentlicht. Da Java-Version 7 erst am 28.07.2011 veröffentlicht wurde, kann man daraus schließen, dass prefuse höchstens mit der Version 6 gebaut wurde. Da Android 2.2.3 Java-Version 6 unterstützt und prefuse höchstens mit Java-Version 6 ent-

32 Quellcode von Android 2.2.3, abgerufen am 18.04.2015 https://android.googlesource.com/platform/frameworks/base/+/android-2.2.3_r2.1/

33 Android Source Tag, abgerufen am 17.08.2015 https://android.googlesource.com/platform/build/+/refs/tags/android-2.2.3_r1

34 Prefuse Download, abgerufen am 18.04.2015 - http://prefuse.org/download/

wickelt werden könnte, werden keine Kompatibilitätsprobleme erwartet. Außerdem findet sich auf einer inoffiziellen Seite[35] der Indiana University Bloomington die Information, dass für das prefuse-Projekt die Mindestanforderung Java 1.4 ist. Dies bestätigt eine Tutorial-Seite[36] von prefuse, auf der steht, dass prefuse mit Java 1.4 geschrieben ist.

Die zweite Quelle, die für AndroidPrefuse benutzt wurde, war OpenJDK[37]. Aus dieser Quelle wurden nur einige Klassen aus dem Paket j*ava.swing* übernommen, da sie in Android 2.2.3 (und auch anderen Android-Versionen) gefehlt haben.

Prefuse verfügt über einige UI-Komponenten wie z.B. *prefuse.util.ui.JrangeSlider,* die auf Swing-Komponenten basieren. Bei der Portierung wurden diese prefuse-Klassen komplett ausgeschaltet. Sie existieren zwar als Klassen, sind aber leer, und der ursprüngliche prefuse-Code befindet sich darin als Kommentar. Die UI-prefuse-Komponenten, die auf Swing-Komponenten basieren, wurden nicht übernommen, weil die UI-Komponenten auf Android komplett anders funktionieren. Deswegen müssen sie in Android neu programmiert werden. Nur einige Nicht-UI-prefuse-Klassen aus dem Paket *prefuse.data.query* sind stark auf einigen Swing-Klassen aufgebaut. Diese Klassen sind Basis-prefuse-Klassen und können nicht ausgeschaltet werden. Sie sind der Grund, warum die ganze Swing-Bibliothek nicht weggeworfen werden konnte. Teilweise benutzen diese Klassen auch UI-Swing-Komponenten, z.B. wie folgende Klasse in Listing 2.

Deswegen wäre wünschenswert, dass man die Klassen aus dem Paket *prefuse.data.query* umprogrammiert, sodass sie nicht mehr von Swing-Klassen abhängig sind. Da dies aufwendig ist, wird es als „Future Work“ markiert und nicht im Rahmen dieser Diplomarbeit durchgeführt.

```
package prefuse.data.query;
public class ListModel extends DefaultListSelectionModel implements
MutableComboBoxModel{
…
}
```

Listing 2: Prefuse-Klasse, die Swing-Klassen vererbt und verwendet

Während der Übernahme von AWT-Klassen wurden einige Klassen angepasst. Die Anpassung orientierte sich daran, so wenige AWT-Klassen wie möglich zu übernehmen. Also wurden nur jene AWT-Klassen übernommen, die für AndroidPrefuse relevant waren. Zusätzlich wurden einige Klassen angepasst, weil darin unbrauchbarer Code vorhanden war. So wurde z.B. die Klasse *awt.java.awt.Font* an Android angepasst. Darin

35 Position Paper by Jeffrey Heer, UC Berkeley, abgerufen am 12.08.2015 http://vw.indiana.edu/ivsi2004/jherr/index.html

36 Prefuse Tutorial – abgerufen am 12.08.2015 http://www.cs.mun.ca/~hoeber/teaching/cs4767/notes/04-prefuse/

37 OpenJDK Quellcode, abgerufen am 17.04.2015 - http://download.java.net/openjdk/jdk7/

wird nichts von der ursprünglichen Font-Klasse verwendet, sondern es wird die Android-Klasse *android.graphics.Typeface* verwendet.

Herzstück der Portierung ist die Klasse *awt.java.awt.AndroidGraphics2D*. Sie ist eine Adapterklasse, die das Mapping der Zeichenfunktionen zwischen AWT *Graphics2D* und der Android-Klasse *Canvas* macht. In Listing 3 sind zwei Methoden aufgeführt, die als Beispiel für dieses Mapping dienen.

```
public void transform(AffineTransform Tx)
{
	this.afineTransform.concatenate(Tx);
	Matrix m = new Matrix();
	m.setValues(createMatrix(Tx));
	canvas.concat(m);
}
public static float[] createMatrix(AffineTransform Tx)
{
	double[] at = new double[9];
	Tx.getMatrix(at);
	float[] f = new float[at.length];
	f[0] = (float) at[0];
	f[1] = (float) at[2];
	f[2] = (float) at[4];
	f[3] = (float) at[1];
	f[4] = (float) at[3];
	f[5] = (float) at[5];
	f[6] = 0;
	f[7] = 0;
	f[8] = 1;
	return f;
}
```

Listing 3: Transformierungsmethoden bei der Adapter-Klasse Android2Graphics

Die Methode „transform“ übernimmt eine AffineTransform (eine AWT-Klasse), erstellt anhand der Methode createMatrix eine *android.graphics.Matrix* und übergibt zur Transformierung diese Matrix an *android.graphics.Canvas*.

Die Klasse *AndroidGraphics2D* ist eine Adapterklasse, die die Inkompatibilitäten bei den Zeichenoperation zwischen AWT und Android löst. Einige Methoden wie z.B. *drawImage* wurden nicht implementiert, da diese für die Implementierung des Prototypbeispiels (Streudiagramm) nicht notwendig waren.

In den folgenden Unterkapiteln wird erläutert, welche Teile modifiziert bzw. neu gemacht wurden und welche Teile noch modifiziert bzw. neu gemacht werden müssen. Die Teile, die übernommen wurden, werden nicht erwähnt, da sie sich automatisch aus den anderen Listen ergeben. Einige Teile von prefuse wurden nicht auf ihre Funktionalität hin geprüft, wie z.B. das Paket *prefuse.data.io*. Dieses wurde nicht verändert, aber da es in

der Beispielimplementierung nicht verwendet wurde, kann keine Aussage darüber gemacht werden, ob es funktionieren würde. Da AndroidPrefuse und prefuse aber dieselbe Bibliothek (*java.io*) verwenden, kann davon ausgegangen werden, dass dieses Paket funktionsfähig ist.

4.2.1 Teile, die teilweise übernommen wurden

Umbenennung von Paketen: Android verbietet den Paketnamen *java.awt*. Daher wurde das Paket in *awt.java.awt* umbenannt.

Umbenennung von Klassennamen: Einige Klassennamen sind schon in Android vorhanden. Obwohl diese Klassen in zwei unterschiedlichen Paketen vorhanden sind, hat Android gemeldet, dass es wegen der Namen Kompatibilitätsprobleme gibt. Deswegen wurden einige Klassen umbenannt. Bsp.: *prefuse.Display → prefuse.PDisplay*, *prefuse.activity.Activity → prefuse.activity.Pactivity, awt.java.awt.Event → awt.java.awt.AWTEvent,* usw.

prefuse.PDisplay: In prefuse erbt die Klasse *Display* von *javax.swing.JComponent*. In AndroidPrefuse erbt sie von der Klasse *android.view.View*. Somit ist es technisch gewährleistet, dass AndroidPrefuse mehrere *Views* gleichzeitig in einer Applikation verwalten kann. Dies ist für die Implementierung des Camera Design-Patterns (siehe Kapitel 3.6.2.1) notwendig. Außerdem haben *JComponent* und *View* eine ähnliche Struktur, z.B. gibt es in *JComponent* die Methode *paintComponent(Graphics g)*; in *View* gibt es das entsprechende Pendant dazu, nämlich *onDraw(Canvas canvas)*.

Die Methoden „zoom“, „pan“, „rotate“, u.a. wurden nicht verändert.

Der Teil, der sich um die Benutzerinteraktionen gekümmert und die Events an die entsprechenden *EventHandlers* weitergeleitet hat, musste neu implementiert werden. Das wurde aus dem Grund gemacht, weil die Benutzerinteraktionen in Android anders als in java-Desktop sind.

Die Methode „*disableHardwareAcceleration*“ wurde hinzugefügt, damit die Hardwarebeschleunigung ausgeschaltet werden kann.

In dieser Klasse wurde auch die Möglichkeit eingebaut, dass die Einträge in unterschiedlichen Threads abgearbeitet werden. In Listing 11 von Kapitel 4.5.2 ist der Code aufgeführt, in dem das Rendering in verschiedenen Threads abgearbeitet wird. Die Subklasse *PDisplay.RenderThread* (Listing 4) wurde dabei neu erstellt.

```
	class RenderThread extends Thread {
		AndroidGraphics2D g;
		int threadNr;
		public RenderThread(AndroidGraphics2D g, int threadNr){
			this.g = g;
			this.threadNr = threadNr;
		}
		public void setThreadNr(int threadNr){
			this.threadNr = threadNr;
		}
		public void run(){
			int from = threadNr * m_queue.rsize/numberThreads ;
			int to = (threadNr + 1) * m_queue.rsize/numberThreads ;
			for (int i = from; i < to; ++i)                {
				m_queue.ritems[i].render(g);
			}
		}
	}
```

Listing 4: Klasse RenderThread bearbeitet die Einträge in eigenem Thread

4.2.2 Teile, die komplett neu gemacht wurden bzw. werden müssen

Color-Klasse: Die Implementierung von Farben in Android und in *java.awt.Color* sind anders. Die Klasse *android.graphics.Paint* repräsentiert die Farbe über einen Integer-Wert. Dagegen gibt es in AWT für die Farbe eine spezielle Klasse, *Color*. Deswegen wurde eine Mapping-Methode geschrieben, die die Transformierung übernimmt (siehe Listing 5).

```
public class AndroidGraphics2D implements Graphics2D {
	...
	public void setColor(Color c)
	{
		this.currentPaint.setColor(c.getAndroidColorRepresentation());
	}
	...
}
public class Color implements PPaint, Serializable {
   ...
   public int getAndroidColorRepresentation()
    {
      return android.graphics.Color.argb(this.getAlpha(),this.getRed(),
this.getGreen(), this.getBlue());

    }
   ...
}
```

Listing 5: Mapping der Farbe von AWT zu Android

Weggeworfene UI-Komponenten: Die UI-Komponenten, die auf Swing basieren, wurden weggelassen und müssen neu implementiert werden. Die meisten Klassen im Paket

prefuse.util.ui wie *JRangeSlider*, *JPrefuseTable*, *JforcePanel,* etc. wurden ebenfalls weggelassen und müssen neu implementiert werden.

awt.java.awt.Font und awt.java.awt.FontMetrics: Die Implementierung dieser Klassen wurde fast ganz ausgeworfen. Der Klasse *FontMetrics* wird eine Instanz von *Font* übergeben. Diese beinhaltet eine Instanz von *android.graphics.Paint*. Die Instanz von p*aint* wird benötigt, um die *FontMetrics* zu liefern.

Controllers: Die Controllers (Komponenten zuständig für die Benutzerinteraktion) mussten neu implementiert werden. Grund dafür sind die komplett unterschiedlichen Bedienkonzepte von Mobilgeräten und Desktop-Computern. Es wurden nur die Controllers implementiert, die für die Implementierung des Streudiagramm-Beispiels notwendig waren. Folgende Controllers wurden implementiert: ToolTipControl (Tooltip), PanControl(Schieben), ZoomControl(Zoomen), ZoomToFitControl (Zoom to fit). In Kapitel 4.5.6 werden diese Bedienkonzepte detaillierter erklärt. In Listing 6 ist ein Teil von ZoomControl dargestellt, um die Implementierung der Controllers zu demonstrieren.

```
public boolean onScale(ScaleGestureDetector detector){
            scaleFactor *= detector.getScaleFactor();

            down = display.getAbsoluteCoordinate(new
Point2D.Float(detector.getFocusX(), detector.getFocusY()), down);

            // don't let the object get too small or too large.
            scaleFactor = Math.max(0.95f, Math.min(scaleFactor, 1.05f));
            Log.d("PZOOM", "scaleFactor: (" + scaleFactor + ")");

            int currentSpan = (int) detector.getCurrentSpan();
            int previousSpan = (int) detector.getPreviousSpan();

            if (display.isTranformInProgress() || Math.abs(currentSpan -
previousSpan ) < 5 ) // if some other transformation is in progress or the
distance between to fingers is not changing (while the fingers remain on
screen)
                  return true;

            zoom(display, down, scaleFactor, true);
            display.invalidate();
            return true;
      }
```

Listing 6: ZoomControl-Implementierung

4.2.3 Verbesserte Teile/Schwachpunkte während der Portierung

Bei der Portierung wurde an einigen Stellen Generizität hinzugefügt. Es gab sehr viele Stellen, an denen dies notwendig war. Weil dies nicht Schwerpunkt dieser Arbeit war, wurden diese Änderungen nicht überall vorgenommen. In Listing 7 ist ein Beispiel für die Einführung der Generizität in AndroidPrefuse zu finden.

```
// prefuse code
Iterator iter = m_focus.entrySet().iterator();
while ( iter.hasNext() ) {
	Map.Entry entry = (Map.Entry)iter.next();
	TupleSet ts = (TupleSet)entry.getValue();
	ts.clear();
}
// AndroidPrefuse code
for (Map.Entry<String, TupleSet> entry : m_focus.entrySet()){
	TupleSet ts = entry.getValue();
	ts.clear();
}
```

Listing 7: Generizität in AndroidPrefuse

4.3 Architektur

Durch die Übernahme der Bibliotheken AWT und Swing unterscheidet sich die Architektur von AndroidPrefuse kaum (oder gar nicht) von der Architektur des originalen prefuses. In Kapitel 3.6.2 wurde die Architektur von prefuse und den darin verwendeten Design-Patterns detailliert beschrieben. Außer dass die Klasse *prefuse.Display* nicht mehr von *Jcomponent,* sondern von *android.view.View* erbt, hat sich bei der prefuse-Struktur, die in Abbildung 30 dargestellt ist, gar nichts verändert. Zusätzlich läuft die Hauptapplikation nicht in Swing-Frame (*javax.swing.Frame*), sondern in einer *android.app.Activity* (für mehr Informationen über Android-Architektur siehe Kapitel 3.4).

AndroidPrefuse implementiert genau wie prefuse das Design-Pattern „Information Visualization Reference Model“ (siehe Abbildung 31). Auch die Sub-Patterns wie z.B. Data Column, Cascaded Tables, Scheduler, Renderer, etc. sind im AndroidPrefuse vorhanden.

4.4 Streudiagramm mit AndroidPrefuse

Das Streudiagramm wurde nach der Anleitung von Herrn Rind [31] erstellt. Bis auf einige Kleinigkeiten ist die Implementierung vom Streudiagramm in AndroidPrefuse dem Beispiel 6[38] aus der Anleitung sehr ähnlich.

Die Methode „*updateBounds*“ ist komplett identisch. Die Methode „*generateTable“* ist strukturell gleich, nur dass anstelle von 3 1.000 Einträge erzeugt werden.

Das implementierte Streudiagramm unterstützt folgende Benutzerinteraktionen: Zoom, Verschieben, Zoom to fit und Tooltip (für die Beschreibung der Benutzerinteraktionen siehe 4.5.6).

In Listing 8 sind die Android-spezifischen Methoden des Streudiagramms aufgeführt. In der Methode *onCreate* wird mittels der Methode „*createVisualization*“ ein *View* erstellt

38 Quellcode von prefuse Scatterplot Implementierung von Alexander Rind, abgerufen am 18.08.2015 - http://www.ifs.tuwien.ac.at/~rind/w/lib/exe/fetch.php/java/scatterplot6.java

und der *android.app.Activity* übergeben. Die Methode „*createVisualization“* hat sich leicht verändert. In Listing 9 sind aus Platzmangel nur die Teile, die sich vom prefuse-Beispiel von Herrn Rind unterscheiden, aufgeführt.

```
public class MainActivity extends Activity{
	Visualization vis;
	PDisplay display;
	@Override
	protected void onCreate(Bundle savedInstanceState){
		super.onCreate(savedInstanceState);
		setContentView(createVisualization(generateTable()));
	}
	protected void onPause() {
	    super.onPause();
	    display.setDamageRedraw(true);
	}
	protected void onResume() {
	    super.onPause();
	    display.setDamageRedraw(true);
	}
}
```

Listing 8: Streudiagramm Implementierung - Android spezifische Methoden

Am Anfang wird eingestellt, dass das Rendering in vier Threads abgearbeitet werden soll. Danach wird die Hardwarebeschleunigung ausgeschaltet, da es Antialiasing-Probleme (siehe Kapitel 4.5.4) mit der Hardwarebeschleunigung gibt.

Danach wird eine *Action* für das Aufrufen der Methode „*updateBounds*“ erstellt. Ohne die *Action* muss der Aufruf der Methode in „*display.post()*“ laufen, weil erst dann die Zeichenkomponente *Canvas* zur Verfügung steht.

Danach wird eine *ItemAction* für das Verändern der Schriftgröße erstellt.

Der letzte Teil, der als Kommentar eingefügt wurde, soll dazu dienen klarzumachen, dass dieser Teil nicht benötigt wird. Es handelt sich um einen *Listener*, der dazu dient, die Grenzen (*updateBounds*) zu aktualisieren, wenn sich die Applikations- bzw. Fenstergröße verändert. Dies ist in AndroidPrefuse nicht mehr notwendig, weil beim Verändern der Applikationsgröße (z.B. Bildschirm von Quer- auf Hochformat stellen) automatisch *repaint* von *View* durchgeführt wird. Dabei werden dann die *Actions* (darunter auch die neu erstellte *Action* für *updateBounds*) aufgerufen.

In Abbildung 44 ist ein Screenshot der Streudiagramm-Implementierung auf dem Testgerät (Samsung Galaxy S5) zu sehen. Die Daten die für das Streudiagramm die zufällig generiert und sie haben keine Bedeutung. Also die Daten dienen nur zur Demonstrationszwecken für die Streudiagramm-Implementierung.

```
private View createVisualization(Table data){
…
	vis = new Visualization();
	display = new PDisplay(this, vis);
	display.setNumberThreads(4); // create 4 threads to render items
	display.disableHardwareAcceleration(); // disable hardware
acceleration , otherwise some drawing code do not work. e.g. antialiasing on
drawing with Path
…
	ActionList draw = new ActionList();
	// this code was added to eliminate the call of the method
"updateBounds" at display.post()
	draw.add(new Action(){
		@Override
		public void run(double frac)
		{
			PDisplay display = getVisualization().getDisplay(0);
			updateBounds(display, boundsData, boundsLabelsX,
boundsLabelsY);
		}
	});

draw.add(new ItemAction()
	{
		//change the font size of the axis labels
		@Override
		public void process(VisualItem item, double frac)
		{
			if( !item.isInGroup("xlab") && !item.isInGroup("ylab") ) //
process only the items of the axis
				return;
		  Font font = FontLib.getFont("SansSerif",Font.PLAIN,25);
			item.setFont(font);
		}
	});
…
	// react on window resize => not needed in AndroidPrefuse
	/*
	display.addComponentListener(new ComponentAdapter() {
		public void componentResized(ComponentEvent e) {
			updateBounds(display, boundsData, boundsLabelsX,
boundsLabelsY);
			vis.run("update");
		}
	});
	*/
…
}
```

Listing 9: Streudiagramm Implementierung – Methode „createVisualization“

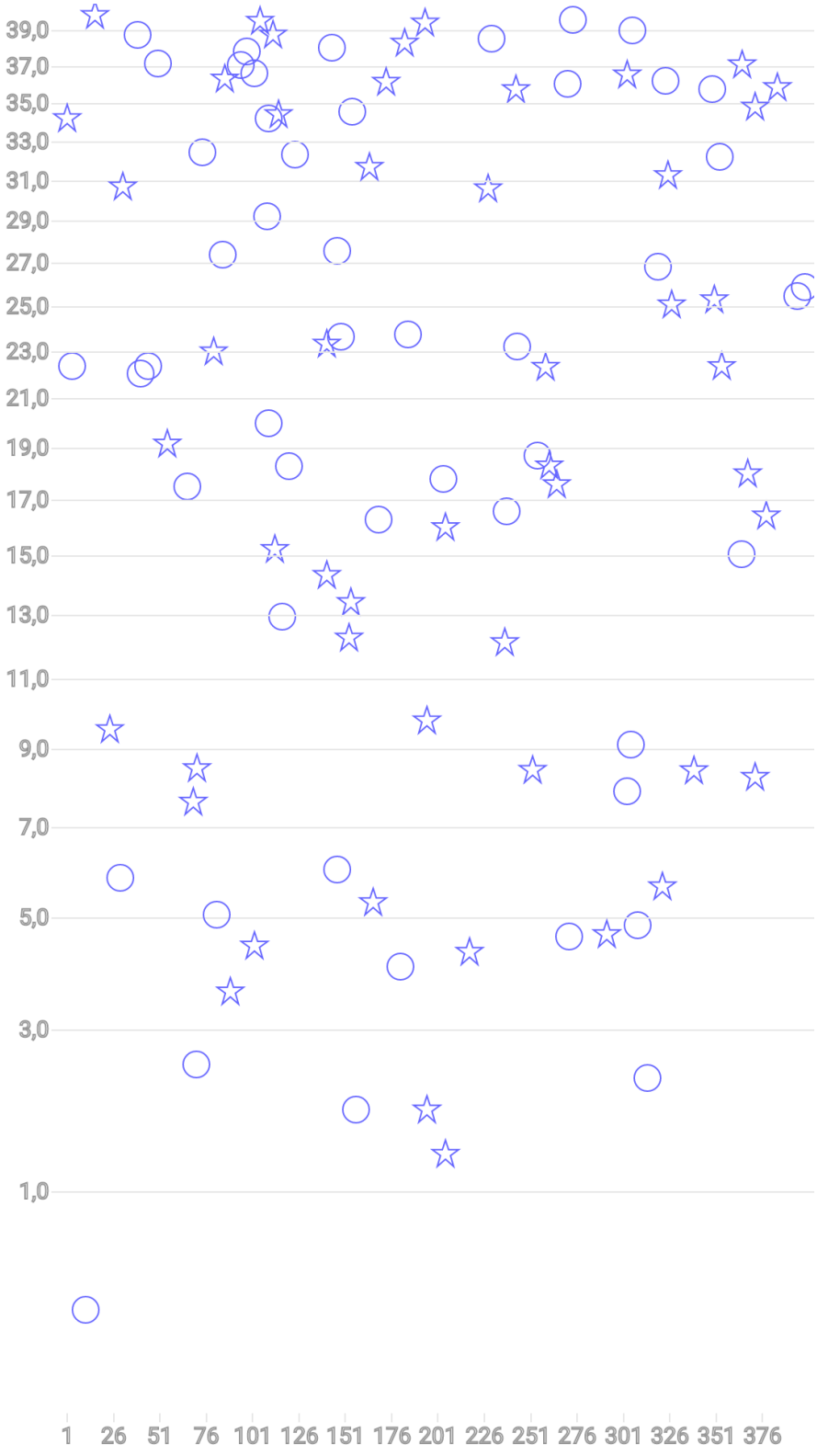

Abbildung 44: Screenshot der Streudiagramm-Implementierung mit AndroidPrefuse (Die Daten für das Streudiagramm sind zufällig generiert)

4.5 Herausforderungen und deren Lösungen

4.5.1 Unterschiede der generellen Architektur zwischen Android und Java-Desktop

Problembeschreibung

Obwohl die Android-Applikationen in der Programmiersprache Java geschrieben werden, können viele Desktop-Applikationen, die in Java geschrieben wurden, nicht eins zu eins in Android übernommen werden. Grund dafür ist, dass auf Android-Geräten keine Java Virtual Machine (von Oracle), sondern die für Android spezifischen Laufzeitumgebungen Dalvik bzw. ART laufen. Diese Laufzeitumgebungen unterstützen viele Softwarebibliotheken wie Swing und AWT, die in Java für Desktop-Applikationen zur Verfügung stehen, nicht. prefuse ist großteils mit Swing und AWT gebaut, weshalb diese Bibliotheken stark und tief in prefuse integriert sind. Dies erschwert die Portierung von prefuse nach Android erheblich.

Lösung

Wie bereits erklärt, ist die Architektur dieser beiden Technologien sehr unterschiedlich. Das größte Problem stellen die fehlenden Bibliotheken wie Swing und AWT dar. Die Bibliothek java.awt wurde 255 Mal (50 eindeutige Klassen) in 97 prefuse-Klassen importiert (geladen) und javax.swing 120 Mal (62 eindeutige Klassen) in 35 prefuse-Klassen importiert (geladen). Aus dieser Aufzählung wurden die Demo- und Testklassen herausgenommen. Davon ausgehend, und mithilfe einer Analyse des Quellcodes wurde festgestellt, dass es folgende Möglichkeiten zur Portierung von prefuse gibt:

1. **Komplette Neuprogrammierung von prefuse**. Dies ist notwendig wegen der nicht vorhandenen und nicht mit Android kompatiblen Swing- und AWT-Klassen. Man könnte vielleicht einen kleinen Teil ohne Veränderung komplett übernehmen. Aber allein das Ermitteln der Teile, die mit Android kompatibel sind, wäre ein großer Aufwand. Das Einzige, was man von prefuse übernehmen könnte, wäre die Struktur, d.h. die verwendeten Design-Patterns.
2. **Übernahme der quelloffenen Swing- und AWT-Klassen**. Bei diesem Ansatz wurde überlegt, den Quellcode der Klassen, die von prefuse verwendet werden, von den oben erwähnten Bibliotheken zu übernehmen. Die Übernahme dieser Klassen sollte kein Problem darstellen, da ihr Code offen ist.

Es wurde entschieden, die zweite Variante für diese Diplomarbeit durchzuführen, da die erste Option zu viel Aufwand und nur eine Nachimplementierung von prefuse bedeuten würde.

In Kapitel 4.2 wurde beschrieben, wie die Portierung durch diese Variante durchgeführt wurde.

4.5.2 Performance-Probleme

Problembeschreibung

Die Performance der Softwareanwendungen, also ihre Reaktionszeit, ist ein wichtiger Bestandteil der Softwareanwendungen. Nielsen [28] beschreibt, dass die maximale Grenze der Reaktionszeit einer Anwendung, ohne dass der Benutzer sie negativ spürt, bei einer Sekunde liegt.

> „**1.0 second** is about the limit for the **user's flow of thought** to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.“ [28]

InfoViz-Applikationen sind im Allgemeinen sehr rechenintensiv, besonders wenn es darum geht, viele Daten zu verarbeiten und darzustellen. Die Ressourcen bei mobilen Geräten sind im Vergleich zu Desktop-PCs stark begrenzt. Deswegen war bereits im Voraus mit Performance-Problemen zu rechnen.

Durch die Übernahme der vielen AWT-Klassen wurde befürchtet, dass diese Klassen ein Overhead erzeugen könnten. Dies wäre allerdings im Speicherbereich zu spüren. Auch in der CPU-Leistung wurden Einbußen wegen der Umrechnung/Umwandlung der Zeichenfunktionen erwartet. Durch die Adapterklasse *AndroidGraphics2D*, die das Mapping der Zeichenfunktionen zwischen AWT *Graphics2D* und der Android-Klasse *Canvas* macht, wurde befürchtet, dass es Performance-Probleme wegen der doppelten grafischen Berechnungen geben würde.

Nach einem Testdurchlauf bestätigte sich, dass die Implementierung des Streudiagramms Performance-Probleme hat. Bei 10.000 Einträgen (Punkten) beträgt die Reaktionszeit über eine Sekunde und die Applikation ist nicht mehr intuitiv nutzbar. In Tabelle 3 und Abbildung 45 sind die gemessenen Werte dargestellt.

Einträge	Erste Ausführung (ms)	Aktualisieren (ms)	Kommentar
500	344	75	Applikation läuft flüssig
1.000	435	152	Applikation läuft flüssig
2.500	757	307	Kleine Störungen
5.000	1.290	729	Größere Störungen
10.000	2.521	1.203	Nicht mehr intuitiv nutzbar

Tabelle 3: Performance Analyse

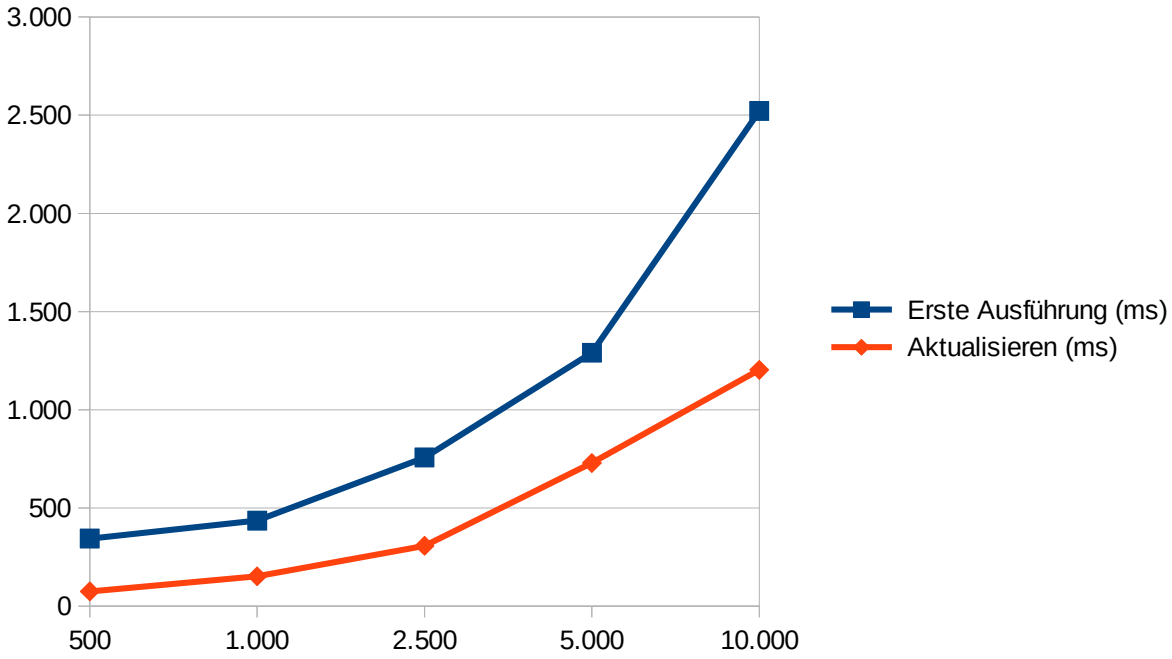

Abbildung 45: Performance-Analyse

Als Testgerät diente ein Samsung Galaxy S5. Dieses hat 2 GB RAM und einen Qualcomm Snapdragon 801 Prozessor, bei dem vier Kerne mit jeweils 2.5 GHz laufen. Als GPU ist Adreno 330 integriert, das mit 578 MHz läuft.

In Tabelle 4 und der Abbildung 46 sind die Ergebnisse der Performance-Analyse von Desktop-Prefuse aufgeführt. Die Applikation erreicht ähnliche Werte wie AndroidPrefuse bei 10.000 Einträgen für das Neuladen der Grafik erst bei 80.000 Einträgen. Daraus ergibt sich ein ein 1:8 Verhältnis. Auf den ersten Blick erscheint dies wie ein großer Unterschied. Wenn man jedoch die großen Hardwareunterschiede bedenkt, ist er gar nicht so schwerwiegend. Nichtsdestotrotz ist die Performance von AndroidPrefuse nicht ideal und erfordert eine Optimierung. Der Performace-Test von Desktop-Prefuse wurde mit einem PC durchgeführt, der folgende Merkmale hat:

- Betriebssystem: Windows 7, Home Edition, 64 Bit
- CPU: Intel i7-3610QM, Quad-Core 2,30GHz
- GPU: AMD Radeon™ HD 7730M Graphics, Core Clock 575 - 675 MHz
- RAM: 8 GB
- HDD: Samsung SSD 840 Pro Series

Einträge	Erste Ausführung Desktop-Prefuse	Aktualisieren Desktop-Prefuse	Vorbereitung der Einträge(ms)	Rendering Der Einträge(ms)	Kommentar
500	491	10	0	10	
1.000	513	16	1	15	Applikation läuft flüssig
2.500	549	39	2	36	Applikation läuft flüssig
5.000	595	74	4	68	Applikation läuft flüssig
10.000	680	144	9	135	Applikation läuft flüssig
20.000	868	291	15	274	Applikation läuft flüssig
30.000	938	452	23	427	Kleine Störungen
40.000	1.194	570	32	536	Größere Störungen
50.000	1.322	719	40	719	Nicht mehr intuitiv nutzbar
70.000	1.796	988	54	931	Nicht mehr intuitiv nutzbar
80.000	1.924	1.152	63	1.085	Nicht mehr intuitiv nutzbar

Tabelle 4: Performance-Analyse von Desktop-Prefuse

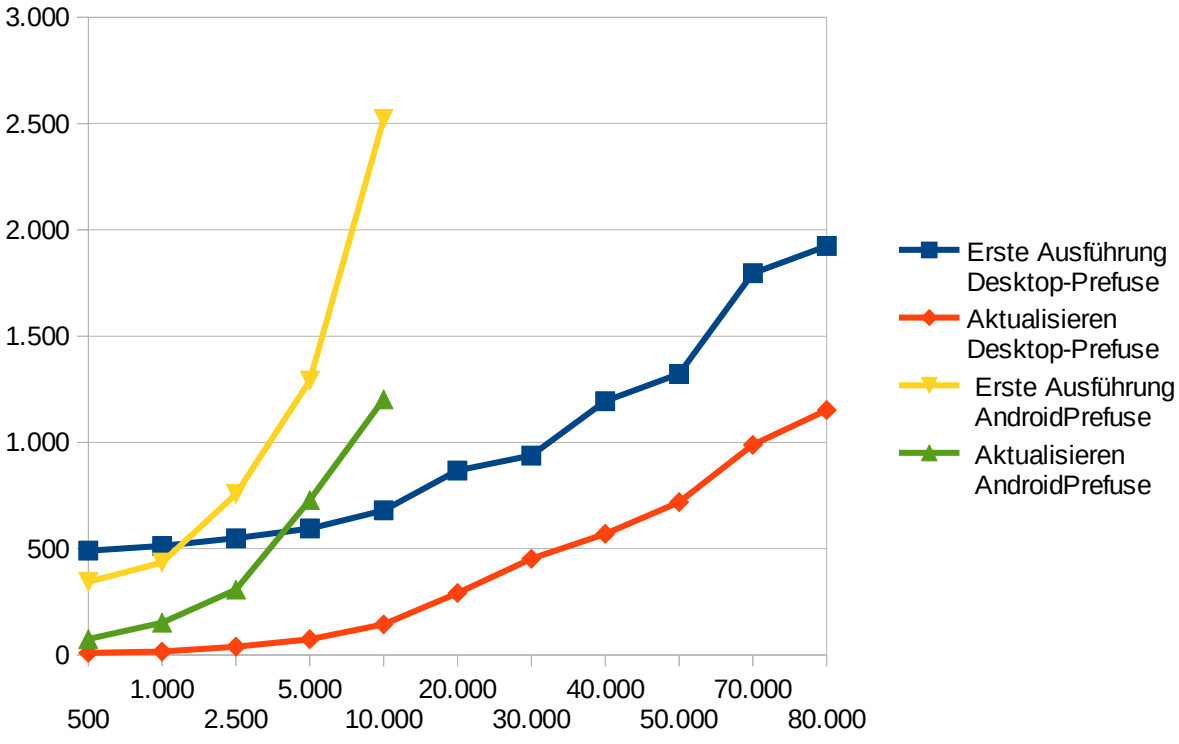

Abbildung 46: Performance-Analyse von Desktop-Prefuse im Vergleich zu AndroidPrefuse

Im Folgenden werden die genaueren Analysen, Tests und Maßnahmen beschrieben, die bei den Versuchen, dieses Problem zu lösen, in Angriff genommen wurden.

Lösung

Um das obengenannte Problem zu lösen, wurden eine Reihe von Tests durchgeführt, um festzustellen, wodurch genau es verursacht wird. Bei keinem der Tests wurden Speicherprobleme, wegen derer die Applikation wegen vielen AWT-Klassen abstürzt, festgestellt. Die Annahme, dass es wegen der vielen AWT- und Swing-Klassen zu einem Overhead kommen könnte, wurde damit widerlegt.

Am Anfang konnte nicht direkt festgestellt werden, welcher Teil genau die Probleme verursacht. Man konnte nur feststellen, dass der Teil, in dem das Rendering der Items gemacht wird (siehe Listing 10), die meiste Ladezeit benötigt. Dieser Teil braucht 84% der Zeit, genauer gesagt 1.056 ms von 1.265 ms. 200 ms braucht der Teil, der berechnet, ob die Einträge sich innerhalb des Sichtbereichs befinden, also in dem Code-Teil, in dem kalkuliert wird, ob der Eintrag zu rendern ist oder nicht.

In Tabelle 5 sind die gemessenen Werte der ersten Analyse dargelegt. Die Spalte „Erste Ausführung“ beinhaltet die Gesamtzeit, die die Applikation beim ersten Start benötigt. Die Spalte „Durchführung der Actions“ beschreibt die Zeit, die die Actions bei der ersten Ausführung brauchen. Die Spalte „Aktualisieren“ beschreibt die Zeit, die die Applikation nach einer Benutzerinteraktion für die erneute Berechnung/Darstellung der Grafik braucht. Die Spalte „Vorbereitung der Einträge“ zeigt die Zeit, die die Applikation benötigt, um für alle Einträge zu berechnen, ob sie gerendert werden sollen oder nicht. Bei der Spalte „Rendering von Einträgen“ sind die Zeiten für das Rendering der Einträge dargestellt.

```
for ( int i=0; i<m_queue.rsize; ++i ) {
    m_queue.ritems[i].render(g2D);
}
```

Listing 10: Rendering der Einträge – Pdisplay.onDraw()

Erste Ausführung	Durchführung der Action (ms)	Aktualisieren (ms)	Vorbereitung der Einträge (ms)	Rendering der Einträge (ms)
332	97	77	10	63
444	149	182	34	144
850	318	334	45	284
1.480	578	621	94	519
2.694	1.128	1.265	201	1.056

Tabelle 5: Erste Performance-Analyse

Nach einer kleinen Analyse fiel auf, dass bei der Methode *getPath(Shape s)* der Klasse *AndroidGraphics2D* jedes Mal eine neue Instanz von dem Objekt Path erstellt wird. Dies zwingt den Garbage Collector, öfters zu laden und umgeht das Caching und die Optimierungen in der Hardware-Pipeline. Da es nur eine Instanz der Klasse AndroidGraphics2D gibt, wurde entschieden, eine Klassenvariable von Typ *Path* zu erstellen und bei der

Methode *getPath* zu verwenden. Dabei wird aber die Methode *Path.rewind*() angewendet, die die vorherigen Linien und Kurven zurücksetzt. Am Ende wurde die Zeit gemessen. Leider wurde mit diesem Vorgehen kein großer Performance-Gewinn erreicht. Die Werte sind in Tabelle 6 eingetragen.

Einträge	Erste Ausführung	Durchführung der Action (ms)	Aktualisieren (ms)	Vorbereitung der Einträge (ms)	Rendering der Einträge (ms)
500	338	119	75	9	61
1.000	472	177	135	17	113
2.500	825	335	370	43	322
5.000	1.473	624	620	90	525
10.000	2.739	1.143	1.187	178	1.001

Tabelle 6 Performance-Analyse nach der Verbesserung von AndroidGraphics2D. getPath() durch nur eine Instanz von Path.

In prefuse-Desktop wurden die Einträge aus Performancegründen zuerst in einem BufferedImage gezeichnet und am Ende wurde das fertige Image am Bildschirm gerendert. In der ersten Version von AndroidPrefuse wurde dieses Verfahren nicht implementiert. Wegen der Performanceprobleme bot es sich an, dieses Verfahren zu implementieren. Nach der Implementierung wurden erneute Tests durchgeführt. Es wurde festgestellt, dass die Applikation dadurch sogar ein bisschen langsamer war. Die Werte dieses Durchlaufs finden sich in Tabelle 7.

Einträge	Erste Ausführung	Durchführung der Action (ms)	Aktualisieren (ms)	Vorbereitung der Einträge (ms)	Rendering der Einträge (ms)	Rendering ohne Zeichenfunktion	Rendering ohne Zeichenfunktion und ohne getPath()
500	347	101	76	10	55	38	37
1.000	449	161	179	30	132	67	62
2.500	833	328	460	63	369	186	146
5.000	1.524	612	721	110	587	323	281
10.000	2.706	1.140	1.444	222	1.180	678	566

Tabelle 7: Performance-Analyse von AndroidPrefuse mit BufferedImage

Bei diesem Test wurde zusätzlich versucht, genauer zu ermitteln, welche Teile wie viel Zeit brauchen. In AndroidGraphics2D wurden die Methodenaufrufe (wie z.B. *canvas.drawRect()*, *canvas.drawPath()* usw.), die das Zeichnen im Android durchführen, weggelassen. Ziel war es, zu sehen, wie viel Zeit diese Zeichenfunktionen brauchen, denn diese Zeit lässt sich nicht optimieren. Die Ergebnisse sind in Tabelle 7 in der Spalte „Rendering ohne Zeichenfunktionen“ dargestellt. Es wurde festgestellt, dass das Rendering bei 10.000 Punkten 678 ms, also 57,4% der Ladezeit benötigt. Die Android-Zeichenoperationen haben also 502 ms (1.180 ms – 678 ms) und damit 42,5% der Zeit gebraucht.

Wie bereits oben erwähnt, kann diese Zeit nicht optimiert werden, also bleibt nur, zu versuchen, den Teil, der 57,4% der Zeit benötigt, zu optimieren. In diesem Durchgang wurde zusätzlich der Aufruf der Methode AndroidGraphics2D.getPath() ausgeschaltet, um zu überprüfen, wie lange der Aufruf dieser Methode dauert. In der Spalte „Rendering ohne Zeichenfunktionen und ohne getPath()“ sind die gemessenen Werte aufgeführt. Dabei ist festzustellen, dass der Aufruf dieser Methode 112 ms (678 ms – 566 ms) dauert. Dieser Methodenaufruf allein nimmt also ca. 9,5% der Renderingzeit in Anspruch. Die Methode *getPath* wandelt den *java.awt.geom.GeneralPath* in *android.graphics.Path* um. Damit wurde die Annahme bestätigt, dass die doppelten graphischen Berechnungen bei Umrechnung/Umwandlung der Zeichenfunktionen von java-Desktop nach Android etwas zum Performance-Problem beitragen.

Da es Probleme (siehe Kapitel 4.5.2) mit der Darstellung von Einträgen, die mit *Path* gezeichnet wurden, gab, wurde die Hardwarebeschleunigung ausgeschaltet. Aus experimentellen Gründen wurde die Hardwarebeschleunigung eingeschaltet, um zu prüfen, ob die Applikation schneller läuft. In Tabelle 8 sind die gemessenen Werte aufgeführt.

Einträge	Erste Ausführung	Durchführung der Action (ms)	Aktualisieren (ms)	Vorbereitung der Einträge (ms)	Rendering der Einträge (ms)
500	344	91	75	14	59
1.000	435	148	152	26	120
2.500	757	317	307	59	244
5.000	1.290	587	729	218	569
10.000	2.521	1.172	1.203	247	931

Tabelle 8: Performance-Analyse mit eingeschalteter Hardwarebeschleunigung

Dabei stellt man fest, dass die bisherigen Maßnahmen zu keiner relevanten Verbesserung geführt haben. Dies kann man nicht als Verbesserung aus visueller Sicht betrachten. Es bleibt dem Programmierer/der Programmiererin von Grafiken überlassen, zu entscheiden, ob er/sie bei seiner/ihrer Applikation Antialiasing beim Zoomen braucht oder nicht. Falls diese Funktion nicht benötigt wird, ist zumindest eine kleine Verbesserung zu erreichen.

Beim nächsten Versuch zur Performance-Verbesserung wurde die Klasse *PDisplay* verändert, indem sie nicht mehr von *android.view.View* erbt, sondern von *android.view.SurfaceView*. Dabei geht es darum, die grafischen Berechnungen für das Rendering und das Rendering selbst nicht mehr in dem Haupt-Thread der Applikation zu machen, sondern in einem eigenen Thread. Dies soll bessere Benutzerinteraktion bieten, da nur die Benutzerinteraktionen im Haupt-Thread verarbeitet und die aufwendigen Berechnungen in einem anderen Thread kalkuliert werden. Die Ergebnisse dieses Tests sind in Tabelle 9 dargestellt.

Erste Ausführung	Durchführung der Action (ms)	Aktualisieren (ms)	Vorbereitung der Einträge (ms)	Rendering der Einträge (ms)
512	105	148	20	92
616	157	162	21	123
1.108	310	364	46	288
1.698	578	643	91	531
3.064	1.209	1.340	181	1.126

Tabelle 9: Performance-Analyse der Applikation mit SurfaceView

Die Werte in der Spalte „Erste Ausführung“ sind nicht aussagekräftig, weil es einen Fehler in der Applikation bei der ersten Ausführung gab. Um diesen Fehler zu umgehen, musste der Code „*sleep(500);*“ eingebaut werden. Dies verzögerte die Durchführung der Applikation bei der ersten Ausführung um 500 ms. Hauptziel für die Performance-Verbesserung ist, die Zeit beim neu Laden nach Benutzerinteraktionen zu verbessern. Aber auch das neu Laden wurde mit der Umstellung auf SurfaceView nicht verbessert. Es wurde sogar schlechter.

Als letzter durchgeführter Versuch zur Performance-Verbesserung wurde entschieden, die Applikation so umzubauen, dass das Rendering aller Einträge in mehreren Threads abgearbeitet wird.

In Listing 11 ist der Code zu finden, der die Threads für das Rendering erzeugt. Es wurde für jeden Thread eine AndroidGraphics2D mit demselben Canvas erstellt. Diese mehrmalige Erstellung von AndroidGraphics2D-Instanzen ist notwendig, weil es in AndroidGraphics2D interne Klassenvariablen gibt, die Werte intern zwischenspeichern. Wenn alle Threads nur eine AndroidGraphics2D-Instanz verwenden, können sie sich gegenseitig stören, z.B. die Farbe der Einträge könnte anders als erwartet ausfallen. Man könnte die Variablen und Methoden synchronisieren. Dann gäbe es aber keinen Performance-Vorteil gegenüber der Lösung ohne Threads, denn dann müssten die Threads bei jeder Zeichenoperation aufeinander warten und würden sequentiell abgearbeitet werden.

```
Thread t ;
for (int i = 0; i < numberThreads; i++){
        AndroidGraphics2D gThread = new AndroidGraphics2D(g2D.getCanvas(), this);
        setRenderingHints(gThread);
        t = new RenderThread( gThread, i );
        t.start();
        threads.add(t);
}
```

Listing 11: Rendering in Threads in PDsiplay.paintDisplay()

In Listing 12 ist die Methode „run“ der Klasse RenderThread dargestellt. Diese berechnet zuerst, welche Einträge bei dem jeweiligen Thread abzuarbeiten sind. Dabei werden die

Einträge gleichmäßig auf die Threads verteilt, je nachdem, wie viele Einträge in den Threads abgearbeitet werden sollen.

```
public void run(){
	int from = threadNr * m_queue.rsize/numberThreads ;
	int to = (threadNr + 1) * m_queue.rsize/numberThreads ;
	for (int i = from; i < to; ++i){
		m_queue.ritems[i].render(g);
	}
}
```

Listing 12: Rendering der Einträge in der Klasse RenderThread

Das Ergebnis des Tests mit dieser Adaption ist in Tabelle 10 dargestellt.

Hier wurde zum ersten Mal eine große Verbesserung gemerkt. Beim neu Laden mit 10.000 Einträgen hat die Applikation nur noch 671 ms gebraucht. Im Vergleich zu vorher (siehe Tabelle 5) wurde die Applikation um ca. 50% verbessert. Bei dieser Lösung muss man aber bedenken, dass es eine Einschränkung gibt.

Einträge	Erste Ausführung	Durchführung der Action (ms)	Aktualisieren (ms)	Vorbereitung der Einträge (ms)	Rendering der Einträge (ms)
500	320	100	83	15	52
1.000	381	157	137	53	61
2.500	674	334	191	44	132
5.000	1.114	588	293	85	173
10.000	2.067	1.138	671	173	477

Tabelle 10: Performance-Analyse der Applikation mit 4 RenderThreads

Das Abarbeiten der Einträge in unterschiedlichen Threads funktioniert nur dann, wenn die Einträge in Bezug auf die Darstellung komplett unabhängig von einander sind. Im Fall des Streudiagramms, bei dem die Einträge ohne Rücksicht auf die anderen gezeichnet werden können, gilt diese Einschränkung nicht. Allerdings musste das Zeichnen der Linien (für die Achsen) synchronisiert werden, weil die Linieneinträge abhängig voneinander waren. Da nur wenige Linien für die Achsen zu zeichnen sind, spielt dies keine Rolle in Bezug auf die Performance. Also obliegt es dem Programmierer/der Programmiererin, zu entscheiden, ob er/sie einen Thread oder mehrere Threads für seine/ihre Grafik benutzt. Die Einstellung der Threads erfolgt ganz einfach: Für vier Threads muss man z.B. nur die Zeile einfügen „*display.setNumberThreads(4);*“ einfügen. Standardmäßig ist nur ein Thread eingestellt, wenn die Methode *setNumberThreads* nicht aufgerufen wird.

Mit dieser Veränderung wurden die Versuche zur Performance-Verbesserung eingestellt. Im Folgenden werden einige Ideen vorgestellt, die zur Perforamce-Verbesserung beitra-

gen könnten. Diese werden aber im Rahmen dieser Arbeit aus Zeitgründen nicht umgesetzt.

Path anstatt GeneralPath verwenden – *android.graphics.Path* soll anstatt von *awt.java.awt.geom.GeneralPath* verwendet werden. Dies erspart die doppelte Transformierung des Paths. Dadurch ist die Methode *AndroidGraphics2D.getPath* nicht mehr notwendig. Wie in einem der früheren Tests (siehe Tabelle 7) festgestellt wurde, benötigt der Aufruf der Methode getPath bei 10.000 Einträgen 112 ms bzw. ca. 9,5% der Renderingszeit. Diese Zeit und eventuell mehr könnte man mit dieser Umstellung einsparen.

Implementierung mit OpenGL ES – OpenGL ES wird verwendet, wenn aufwendige grafische Applikationen, wie z.B. Spiele, 3D Visualisierungen erstellt werden sollen. Es soll performanter als die normale Programmierung mit Canvas sein, wenn es darum geht mehrere Objekte zu verwalten[30]. Die Bedienung der Streudiagramme (und den meisten der InfoVis-Applikationen) mit Benutzerinteraktionen wie z.B. Verschieben und Zoomen basiert auf dem selben Prinzip wie bei einem Spiel. Es wäre also naheliegend, die Implementierung des Kerns von AndroidPrefuse mit OpenGL ES zu machen. Dies soll jedoch nur ein Ziel für die Zukunft von AndroidPrefuse sein und noch nicht für dieser Arbeit umgesetzt werden. Dafür gibt es drei Gründe:

1. Das Programmieren mit OpenGL ES ist ganz anders als das Programmieren mit Canvas. Daher müsste der Quellcode stark angepasst werden.

2. OpenGL ES soll aufwendig zu erlernen sein.

4.5.3 Lizenzfragen

Problembeschreibung

Ziel und Wunsch dieser Arbeit ist es, den Quellcode unter einer freien Lizenz zu handeln, sodass der Code in jeder Form (verändert und unverändert) für kommerzielle oder nichtkommerzielle Zwecke verwendet werden kann.

Da der Quellcode von Java offen ist, ist eine der möglichen Lösungen für die Portierung von prefuse nach Android die Übernahme dieser Klassen. Google hat bei früheren Android-Versionen (Versionen bis 2.2.3) viele Klassen von AWT in Android aus Apache Harmony übernommen und unter Apache Lizenz Version 2.0 gestellt. Diese Lizenz ist für diese Diplomarbeit am geeignetsten. Allerdings werden zusätzlich auch einige Swing-Klassen benötigt. Diese Klassen waren in keiner der Android-Versionen vorhanden. Also wurden als erstes diese Klassen nur aus OpenJDK[39] übernommen. Der Quellcode von Open JDK ist unter der „GNU General Public License version 2“, kurz GPL2, gestellt. Die Klassen sind Teile der GNU Classpath Bibliotheken[40]. Bei dieser Lizenz ist der Quellcode zwar offen, aber jede Veränderung im Code setzt voraus, dass auch der eigene Code

39 OpenJDK Quellcode, abgerufen am 17.04.2015 - http://download.java.net/openjdk/jdk7/

40 GNU Classpath, abgerufen am 17.04.2015 http://www.gnu.org/software/classpath/faq/faq.html#faq1_1

unter derselben Lizenz veröffentlicht werden muss. Es gibt bei den Klassen von OpenJDK allerdings eine Ausnahme:

> „As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module. An independent module is a module which is not derived from or based on this library.“ [41]

Aufgrund dieser Ausnahme könnte man also aus den benötigten Klassen ein jar-Paket erstellen, ohne deswegen das Projekt unter GPL stellen zu müssen. Allerdings erlaubt Android in keinem Fall die Verwendung der Pakete *javax.swing* und der darunterliegenden Pakete, auch wenn sie als jar-Paket geladen werden. Somit ist dieser Versuch gescheitert.

Lösung

Die übernommenen AWT-Klassen aus Android-Version 2.2.3 sind unter Apache Lizenz Version 2 gestellt und stellen kein Problem in dieser Hinsicht dar. Da prefuse einige Swing-Features verwendet, werden auch einige Swing-Klassen benötigt. Der Sourcecode dieser Klassen wurde anfangs von OpenJDK übernommen. Deren Lizenz GNU GPL störte das obengenannte Ziel.

Als Erstes wurde versucht, AndroidPrefuse unabhängig von Swing zu machen. Dabei wurden viele prefuse-Klassen, die Swing UI-Komponenten benutzen, ausgeschaltet. Diese Klassen sind zwar in AndroidPrefuse vorhanden, sie sind aber leer. Die Funktionalität dieser Klassen muss komplett neu implementiert werden. Das Ausschalten dieser prefuse-Klassen (wie z.B. *prefuse.util.ui.JRangeSlider*) erfolgte aus folgenden Gründen:

1. Um die Abhängigkeit von Swing abzuschaffen und damit auch die Abhängigkeit von GNU GPL.
2. Die Swing UI-Komponenten sind nicht kompatibel mit Android UI-Komponenten, weil sie komplett andere Ansätze verfolgen.

Bei einigen Core (nicht komplett UI) prefuse-Klassen (z.B. die Klasse *prefuse.data.query.ListModel*, siehe Listing 2) sind die Swing-Klassen tiefer im Basis-prefuse integriert. Um sich aus der Swing-Abhängigkeit zu lösen, braucht man viel mehr Aufwand und zieht wahrscheinlich größeres Refactoring von AndroidPrefuse mit. Also konnte in erster Instanz die Swing-Abhängigkeit nicht komplett abgeschafft werden.

Die Lizenz von GNU Classpath verfügt über eine Ausnahme [41]. Diese Ausnahme erlaubt die Verwendung dieser Klassen, ohne auf auf die GPL zu achten, wenn man sie

in seinem Projekt verlinkt. Die verwendeten Swing-Klassen wurden kompiliert und dann in AndroidPrefuse verwendet. Android hat die Verwendung der Pakete *java.** und *javax.** verweigert. Also müsste man die Namen der Pakete ändern. Außerdem wurde die Methode *javax.swing.event.EventListenerList.toString()* verändert, damit die Klasse *gnu.java.lang.CPStringBuilder* nicht mitgenommen werden muss. Wenn diese Veränderung nicht gemacht wird, kommen viele weitere Klassen/Pakete wegen der Abhängigkeiten mit. Die Änderung der Paketnamen und die Änderung der *toString*()-Methode stellen allerdings eine Verletzung der GPL Lizenz dar. Zwei Möglichkeiten sind zur Lösung dieses Problem geblieben:

1. Das Stellen von AndroidPrefuse unter GPL Lizenz oder
2. Auflösung von Swing-Klassen in AndroidPrefuse

Da keine dieser Möglichkeiten zufriedenstellend ist, wurde nach weiteren Lösungsansätzen gesucht. Nach einiger Recherche wurde festgestellt, dass das Projekt Apache Harmony[41] auch die Implementierung von Swing-Klassen beinhaltet. Dieses Projekt ist genau wie Android unter Apache Lizenz Version 2 gestellt. Dadurch kann AndroidPrefuse unter der gewünschten Lizenz veröffentlicht werden. Also wurden die Swing-Klassen aus diesem Projekt übernommen und die Klassen von OpenJDK verworfen. Der Code von AndroidPrefuse wurde am 15.09.2015 bei GitHub[42] veröffentlicht. Die README[43] stellt die Rahmenbedienungen für die Lizenz[44] von AndroidPrefuse dar. Da viele Klassen von prefuse übernommen wurden, musste auch die prefuse-Lizenz[45] im Projekt mit veröffentlicht werden.

4.5.4 Antialiasing-Probleme

Problembeschreibung

> „Antialiasing (AA), auch Anti-Aliasing oder Kantenglättung, ist die Verminderung von unerwünschten Effekten, die durch das begrenzte Pixelraster (Bildauflösung) entstehen können (Alias-Effekt mit dem Pixelraster) oder die durch den Treppeneffekt bei der Erzeugung einer Computergrafik (computergenerierte 2D oder 3D Grafiken) entstehen.“ *[42]*

41 Apache Harmony, abgerufen am 19.09.2015 - http://harmony.apache.org/

42 Quellcode von AndroidPrefuse, abgerufen am 19.09.2015 https://github.com/dritanlatifi/AndroidPrefuse

43 Kurze Beschreibung von AndroidPrefuse für GitHub, abgerufen am 19.09.2015 https://github.com/dritanlatifi/AndroidPrefuse/blob/master/README

44 AndroidPrefuse Lizenz, abgerufen am 19.09.2015 https://github.com/dritanlatifi/AndroidPrefuse/blob/master/license-androidprefuse.txt

45 Prefuse-Lizenz, abgerufen am 19.09.2015 https://github.com/dritanlatifi/AndroidPrefuse/blob/master/license-prefuse.txt

Wenn die Hardwarebeschleunigung in Android beim Streudiagramm-Beispiel eingeschaltet wurde, wurden die Einträge, die mit der Klasse *android.graphics.Path* gezeichnet wurden, beim Hineinzoomen (Vergrößern) verpixelt dargestellt. Das Einschalten von Antialiasing in Android hat dabei überhaupt keinen Effekt gezeigt.

Lösung

Das Einschalten von Hardwarebeschleunigung in Android hat beim Zoomen die mit *android.graphics.Path* gezeichneten Einträge nicht mehr scharf dargestellt. Das Einschalten von Antialiasing (siehe Listing 13) hat die Situation auch nicht verbessert.

```
protected Paint currentPaint = new Paint();
…
private void applyHints(){
	...
	currentPaint.setAntiAlias(true);
}
```

Listing 13: Einschalten von Anti Aliasion in Android2Graphics

Nach einer Recherche wurde festgestellt, dass Android momentan (Stand 15.04.2015) das Antialiasing Feature teilweise nicht unterstützt [20]. In Tabelle 11 sind die unterstützen bzw. nicht unterstützten Methoden aufgelistet, und falls sie unterstützt werden, ab welcher Android API-Version dies der Fall ist. Hierbei sieht man, dass unter anderem die Zeichenmethode *drawPath* nicht unterstützt wird.

Damit die Grafik auch nach dem Zoomen scharf dargestellt wird, wurde eine Einstellmöglichkeit in AndroidPrefuse eingebaut, um die Hardwarebeschleunigung ausschalten zu können. Damit wurde die Hardwarebeschleunigung in der Streudiagramm-Implementierung ausgeschaltet. Dies hat keine Performance-Nachteile ergeben (siehe Kapitel 4.5.2).

Drawing operation to be scaled	First supported API level
drawText()	18
drawPosText()	✗
drawTextOnPath()	✗
Simple Shapes*	17
Complex Shapes*	✗
drawPath()	✗
Shadow layer	✗

Tabelle 11: Unterstützte und nicht unterstützte Zeichenmethoden in Android Canvas während Skalierung. Quelle: [20]

***Legende**:*

x-Kein Antialiasing-Support,

Farbe „rot“ - wurde von Autor hinzugefügt und kennzeichnet die Methode, bei der Antialiasing-Probleme aufgetreten sind [20]

4.5.5 ScaleGestures – Probleme

Problembeschreibung

Bei der Implementierung des Zooms traten zwei Probleme auf:

1. Der Fokus des Zooms war beim Zoomen nicht richtig zentriert

2. Die Zoomgeschwindigkeit war zu schnell, z.B. wurde während des Hineinzoomen die Grafik bei kleinen Fingerbewegungen zu sehr vergrößert. Diesbezüglich schreibt Nielsen in [28] folgendes:

> „Normally, response times should be as fast as possible, but it is also possible for the computer to react so fast that the user cannot keep up with the feedback. For example, a scrolling list may move so fast that the user cannot stop it in time for the desired element to remain within the available window.“ [28]

Lösung

1. Nach einer Analyse wurde festgestellt, dass der Fokus auf den ersten Fingertouch gesetzt war. So wurde eine Möglichkeit gesucht, den Fokus zwischen den zwei Fingern zu positionieren. Die Lösung war, den Fokus ständig während der Bewegung der beiden Finger zu aktualisieren. In Listing 14 steht der Code, in dem die Variable „down“, die den Fokus der Skalierung darstellt, in der Methode „onScale“, die bei jeder Bewegung der beiden Finger aufgerufen wird, aktualisiert wird.

```
public boolean onScale(ScaleGestureDetector detector){
  …
  down = display.getAbsoluteCoordinate(new Point2D.Float(detector.getFocusX(),
                detector.getFocusY()), down);
  scaleFactor = Math.max(0.95f, Math.min(scaleFactor, 1.05f));
  …
}
```

Listing 14: Fixieren des Fokus' zwischen zwei Fingern während des Zoomens

2. Für die Lösung dieses Problems wurden die Grenzen zum Skalierfaktor (ScaleFactor) verkleinert. Die Grenzen wurden von 0,9-1,2 auf 0,95-1,05 herabgesetzt (siehe Listing 14).

4.5.6 Interaktionsprobleme – Differenz zwischen Maus und Touch

Problembeschreibung

Bei der Portierung von Android mussten klarerweise die *Controllers* (die Komponenten in prefuse, die für die Benutzerinteraktion zuständig sind) komplett verändert werden. Die Veränderungen waren nötig, weil die Bedienkonzepte von Desktop-PC und mobile Geräte komplett unterschiedlich sind. Beim Desktop werden die Applikationen durch Maus und Tastatur bedient. Bei mobilen Geräten werden hauptsächlich Touch-Gesten

(durch Finger oder Stylus) verwendet, teilweise einige wenige Knöpfe. Die Hauptherausforderung war also, die Mausinteraktionen durch Touchinteraktionen zu ersetzen.

In dem Streudiagramm-Beispiel von Herrn Rind sind folgende Benutzerinteraktionen vorhanden:

- **Schieben** (Drag & Drop) – Hier wird das Schieben durch das Halten der linken Maustaste und gleichzeitige Bewegung der Maus in die gewünschte Richtung erreicht.
- **Tooltip** - Detaillierte Information zu einem Eintrag – Definition:

  „Tooltips are displayed when you roll over an icon with the cursor. It may take a second or two to display the tooltip, but when it does appear, it usually is a small box with a yellow background explaining what the icon represents.“ *[12]*

  Beim Streudiagramm-Beispiel werden mit Tooltip die zwei numerischen Parameter angezeigt.
- **Zoomen** – Beim Streudiagramm-Beispiel wird das Zoomen durchgeführt, indem man die rechte Maustaste gedrückt hält und die Maus nach oben bzw. nach unten für das Hinauszoomen bzw. Hineinzoomen zieht.
- **Zoom to fit –** Mit einem Klick mit der rechten Maustaste irgendwo in der Grafik wird die Größe der Grafik an den Bildschirm angepasst.

Lösung

In dem Streudiagramm-Beispiel wurden die Benutzerinteraktionen wie folgt gelöst:

- **Schieben** – Das Schieben der Grafik wurde hier wie bei jeder Android-Applikation gelöst: Die Grafik wird verschoben, indem man mit einem Finger über den Bildschirm wischt. Beispiel: Wenn man in dem Streudiagramm die Grafik nach links verschieben will, legt man einen Finger auf den Bildschirm und bewegt ihn nach links, ohne den Finger vom Bildschirm zu nehmen.
- **Tooltip** – Detaillierte Informationen zu einem Eintrag werden auf dem Bildschirm angezeigt, indem man kurz mit einem Finger auf den gewünschten Eintrag tippt. Die Informationen werden als Android-Toast-Nachricht (siehe Abbildung 47) angezeigt (eine Android Toast-Nachricht ist ein kleines PopUp-Fenster, das für kurze Zeit auf dem Bildschirm erscheint).
- **Zoomen** – Das Zoomen wird durch die Bewegung zweier Finger durchgeführt. Wenn man die Finger aufeinander zubewegt, wird die Grafik verkleinert. Wenn man die Finger auseinander bewegt, wird die Grafik vergrößert.

- **Zoom to fit –** Mit einem Doppeltipp auf den Bildschirm (zwei Mal schnell hintereinander tippen) wird die Grafik an den Bildschirm angepasst. Die Grafik wird also verkleinert, oder bewegt, damit die gesamte Grafik in den Bildschirm passt.

Abbildung 47: Detaillierte Information mit einer Android Toast-Nachricht

Für jede dieser Benutzerinteraktionen gibt es in Android einen entsprechenden Handler. Diese Handlers wurden dem AndroidPrefuse weitergeleitet. Danach wurden die entsprechenden Controllers in AndroidPrefuse für die jeweiligen Handlers programmiert.

4.5.7 Unterschied in der Displaygröße

Problembeschreibung

Android und besonders Prefuse verwalten die Positionierung der Elemente relativ gut. Da bei mobilen Geräten die Displaygröße viel kleiner als bei Desktop-PCs ist, ist von Anfang an bekannt, dass die Visualisierungen nicht eins zu eins übernommen werden können. Das Testgerät Samsung Galaxy S5 hat eine Bildschirmgröße von 5,1 Zoll und eine Bildschirmauflösung von 1080 x 1920 Pixel. Da die Pixeldichte relativ hoch ist (ca. 432 ppi), werden die Einträge und die Beschriftungen bei den Achsen zu klein angezeigt.

Lösung

Die Lösung dieses Problems war schnell gefunden, da AndroidPrefuse viele Einstellungsmöglichkeiten in dieser Richtung anbietet. In Listing 15 wird mit *ShapeRenderer* die Größe der Einträge auf „20“ gesetzt. In Listing 16 wird mit einer neuen *Action* die Schriftgröße auf „25“ gestellt. Im Originalbeispiel in Desktop-prefuse ist die Eintraggröße auf „7“ und die Schriftgröße auf „12“ gesetzt.

```
vis.setRendererFactory(new RendererFactory()
{
      AbstractShapeRenderer sr = new ShapeRenderer(20);
      Renderer arY = new AxisRenderer(Constants.FAR_LEFT, Constants.CENTER);
      Renderer arX = new AxisRenderer(Constants.CENTER, Constants.FAR_BOTTOM);

      public Renderer getRenderer(VisualItem item)
      {
         return item.isInGroup("ylab") ? arY : item.isInGroup("xlab") ? arX :
sr;
      }
});
```

Listing 15: Einstellung der Größe der Einträge

```
draw.add(new ItemAction()
            {
                  //change the font size of the axis labels
                  @Override
                  public void process(VisualItem item, double frac)
                  {
                        if( !item.isInGroup("xlab") && !
item.isInGroup("ylab") ) // process only the items of the axis
                              return;
                    Font font = FontLib.getFont("SansSerif",Font.PLAIN,25);
                        item.setFont(font);
                  }
            });
```

Listing 16: Einstellung der Schrift (Schriftgröße)

4.6 Vor- und Nachteile von AndroidPrefuse gegenüber vorhandenen Visialisierungsbibliotheken auf Android

In diesem Abschnitt werden die Vor- und Nachteile von AndroidPrefuse gegenüber vorhandenen Visualisierungsbibliotheken auf Android zusammengefasst.

Vorteile:

- Die Flexibilität beim Bauen von neuen und komplexen Visualisierungen

- Wiederverwendungsmöglichkeit von schon vorhandenen Komponenten

- Wartbarkeit, Umschreiben und Erweiterbarkeit von eigenen Visualisierungen, z.B. sind Tausch, Veränderung und Hinzufügen von Benutzerinteraktionen mit wenig Aufwand möglich, da nur die Komponente „Controller“ verändert bzw. neu geschrieben werden muss.

- Leichte Portierung von bereits vorhandenen prefuse-Visualisierungen

Nachteile:

- Zum Teil hohe Lernkurve am Anfang wegen des polylithischen Designs. Anfänger könnten Probleme haben, zu Beginn die Übersicht über die vielen Klassen und Komponenten zu wahren.

4.7 Expert/inneninterview

In Kapitel 4.5 wurden einige Herausforderungen und Probleme in Bezug auf die Visualisierung erläutert. Um zu prüfen, ob sie aus der Sicht von Benutzer/innen implementiert wurden, wird ein Interview mit einer Expertin[46] aus dem Visualisierungs- und Mobilenbereich durchgeführt.

Das Interview fand nicht in Form eines Gesprächs, sondern per E-Mail statt. Dabei wurde die Applikation installationsfähig als Anhang in der E-Mail mitgeschickt. Zusätzlich wurde ein Fragenkatalog zur Verfügung gestellt.

Neben den Fragen wurde der Expertin auch der folgende Text mitgeschickt:

Sehr geehrte Damen und Herren,

anbei ist eine apk-Datei, die eine einfache Android-Applikation installiert. Bei der Applikation handelt es sich um ein einfaches Streudiagramm. Die Daten, die das Streudiagramm abbildet, sind zufällig ausgewählt und haben keine nähere Bedeutung. In meiner Diplomarbeit und auch diesem Interview/Test stehen nicht die Daten, sondern die Implementierung des Streudiagramms im Vordergrund. Unten werden Ihnen diesbezüglich genauere Fragen gestellt.

Vielen Dank, dass Sie sich Zeit dafür nehmen.

Im Folgenden sind die Interviewfragen zusammen mit der Antwort der Expertin und den Bemerkungen vom Autor zu den Antworten aufgeführt:

- Mit welchem Gerät haben Sie die Applikation getestet?
  - **Ziel der Frage**: Im Falle von Performance-Problemen das getestete Gerät besser kennenlernen und die Ermittlung anderer technischer Probleme, die gerätespezifisch sein können.
  - **Antwort der Expertin:** Google Nexus 4
  - **Subfragen**
    - Wenn möglich, technische Daten des Geräts angeben (z.B. Wie viele Prozessoren besitzt es, welche Taktfrequenz besitzen die Prozessoren, wie viel Arbeitsspeicher (RAM) hat das Gerät?)

46 Mit dem Begriff Expertin sind beide Geschlechter gemeint

        - **Antwort der Expertin:** Prozessor: Qualcomm Snapdragon S4 Pro (Quad-core) mit 1,5 GHz, RAM: 2GB
        - Wie groß ist der Bildschirm des getesteten Geräts?
        - **Antwort der Expertin:** 4,7" Bildschirm mit 1280 × 768 Pixel bei 318ppi
        - Welche Android-Version läuft auf dem Gerät?
        - **Antwort der Expertin:** Android 5.1.1
- Haben Sie die Zoom-Aktion benutzt?
    - **Antwort der Expertin:** Ja
    - Sind Sie intuitiv auf die Zoom-Aktion gekommen oder haben Sie Zeit dafür gebraucht?
    - **Antwort der Expertin:** Um ehrlich zu sein, habe ich mir vorher den Fragebogen durchgelesen, sodass ich wusste, dass es eine Zoom-Funktion gibt. Allerdings musste ich ja noch herausfinden, wie es geht – wie gewohnt mit Pinch-to-Zoom. Das war nicht schwer für mich herauszufinden.
    - War die Zoom-Geschwindigkeit für Sie ausreichend schnell?
    - **Antwort der Expertin:** Die Zoomen ist für mich zu schwerfällig, reagiert also zu langsam.
    - **Bemerkung des Autors:** Bei der Entwicklung von AndroidPrefuse gab es Probleme bei der Geschwindigkeit, aber in der umgekehrten Richtung. Das Zoomen war zu schnell. Dies muss näher untersucht werden. Anscheinend ist die Reaktionszeit von AndroidPrefuse beim Zoomen geräteabhängig. Es soll untersucht werden, ob in AndroidPrefuse eine Möglichkeit eingebaut werden kann, damit anhand der aktuellen Framerate dynamisch die Zoomgeschwindigkeit angepasst wird.
    - Haben Sie Bemerkungen zur Zoom-Aktion?
    - **Antwort der Expertin:** Meiner Meinung nach wäre es gut, wenn dieAchsenbeschriftungen immer im Bild zu sehen sind, wenn man reingezoomthat. So hat man keine Verbindung mehr zur Skala und könnte deshalb recht schnell den Überblick über die Daten verlieren. Vor allem beim Auszoomenläuft das Zoomen für mich nicht flüssig genug.
    - **Bemerkung des Autors:** Das Anzeigen der Achsenbeschriftungen in jedem Zustand, auch wenn man reingezoomt hat, ist eine gute Erweiterungsmöglichkeit.
- Haben Sie die Verschieb-Aktion (engl. panning) benutzt?

- **Antwort der Expertin:** Ja
- Sind sie intuitiv auf die Verschieb-Aktion gekommen oder haben Sie Zeit dafür gebraucht?
- **Antwort der Expertin:** Das kam automatisch in Verbindung mit dem Zoomen.
- War die Geschwindigkeit dieser Aktion für Sie ausreichend schnell?
- **Antwort der Expertin:** Auch das Verschieben ist für mich zu schwerfällig – reagiert zu langsam auf meine Interaktion.
- **Bemerkung des Autors:** Anscheinend waren die Optimierungen, die im Kapitel 4.5.2 beschrieben wurden, nicht ausreichend für eine flüssige Darstellung des Streudiagramms. In Zukunft muss weiter untersucht werden, ob die Performance von AndroidPrefuse gesteigert werden kann.
- Haben Sie Bemerkungen zu dieser Aktion?
- **Antwort der Expertin:** Wenn man den Finger zu schnell bewegt, reagiert das Panning oft nicht.
- **Bemerkung des Autors:** Dies konnte nicht reproduziert werden.

- Haben Sie versucht, auf einen Eintrag (Stern oder Kreis) zu tippen?
  - **Antwort der Expertin:** Bis ich zu dieser Frage gekommen, bin noch nicht. Aber ich hätte es sicher ausprobiert.
  - Falls ja, was hatten Sie dabei erwartet?
  - **Antwort der Expertin:** Ich würde erwarten, dass ich Detaildaten zu dem gewählten Datenpunkt bekomme.
  - Sind die angezeigten Information übersichtlich und gut erkennbar?
  - **Antwort der Expertin:** Eine Toast-Nachricht finde ich nicht sonderlich sinnvoll, um Detaildaten von einem Element anzuzeigen, ein Overlay, das weggeklickt werden muss, wäre besser. Vor allem, weil ich als User mit einem Overlay wirklich genug Zeit habe, Detaildaten zu lesen. Mit den Informationen selbst kann ich jetzt nicht viel anfangen. Das liegt aber eher daran, dass ich nicht weiß, um was für einen Datensatz es sich handelt. Ein Problem ist auch, wenn ich die Detailinfos eines Elementes gerade sehe und ein weiteres tappe, dann wird die vorherige Detailinfo fertig angezeigt und erst wesentlich später das zweite getappte Element.
  - **Bemerkung des Autors:** Ein Overlay anstatt einer Toast-Nachricht zu verwenden, ist eine gute Erweiterung.

- Haben Sie Bemerkung zu dieser Funktion?
- **Antwort der Expertin:** Es macht für mich den Eindruck, dass ich nicht alle Elemente antippen kann. Das ist in einem relativ weit ausgezoomten Zustand. Wenn ich total weit eingezoomt bin, dann kann ich alle antippen. Gut wäre es, wenn das gewählte Element selbst auch gehighlighted würde. Zudem wird auch beim Zoomen, wenn man aus Versehen ein Element berührt, die Info angezeigt, was irritierend ist.
- **Bemerkung des Autors:** In einem weit ausgezoomten Zustand ist es normal, dass man nicht den gewünschten Eintrag eintippen kann. Der Expertin kann Recht gegeben werden, dass es in diesem Zustand nicht leicht ist, mit dem Finger einen Eintrag zu treffen. Prefuse (im Desktop) hat in diesem Zustand besser funktioniert, da mit einer Maus besser auf die Einträge gezielt werden kann. Es könnte in Zukunft für AndroidPrefuse erforscht werden, wie und ob Einträge besser mit dem Finger getroffen werden können. Das Anzeigen der Detailinformationen kann statt mit einem einfachen leichten Tippen mit einem langen Fingertipp auf dem Bildschirm implementiert werden. Somit müssen die Benutzer/innen etwas länger auf einen Eintrag tippen, wenn sie Detailinformationen zu dem Eintrag angezeigt bekommen wollen.

- Haben Sie versucht, zweimal hintereinander auf den Bildschirm zu tippen (Doppeltipp)?
  - **Antwort der Expertin:** Bisher noch nicht, darauf wäre ich vermutlich allerdings nicht gekommen.
  - Falls ja, ist das passiert, was Sie erwartet haben? Falls nein, was haben Sie erwartet?
  - **Antwort der Expertin:** Ich hätte erwartet, dass ausgezoomt wird, wenn ich weit eingezoomt bin, wie aus dem Web bekannt. Allerdings würde ich auch erwarten, dass beim Doppeltipp im ausgezoomten Zustand auch wieder eingezoomt wird.
  - **Bemerkung des Autors:** Dass beim Doppeltipp im normalen Zustand eingezoomt wird, ist eine gute Erweiterungsmöglichkeit.
  - Haben Sie Bemerkung zu dieser Funktion?
  - **Antwort der Expertin:** Keine weiteren außer die gerade eben schon. Das Tempo ist angenehm beim Auszoomen.
- Wie lange haben Sie gebraucht, um die ganze Applikation zu testen?
  - **Antwort der Expertin:** Ich habe nebenbei gleich die Fragen beantwortet und es hat ca. 15 Minuten gedauert bis hierhin.

- Waren die Benutzerinteraktionen für die implementierte Funktion geeignet?
  - **Antwort der Expertin:** Ja. Es könnte noch weiter ausgebaut werden, z.B. mit Highlighting des gewählten Elementes, Filter-Funktionen, um die große Menge an Daten besser durchsuchen zu können und eventuell auch fixer Achsenbeschriftungen.
  - **Bemerkung des Autors:** Highlighting der gewählten Einträge, Filter-Funktionen und fixe Achsenbeschriftungen sind gute Erweiterungsmöglichkeiten.
- Wie war die Reaktionszeit der Applikation? War sie zu schnell/zu langsam?
  - **Antwort der Expertin:** Für mich war die Reaktion auf die Gesten für Zooming und Panning zu langsam. Bei Auswählen eines Items war es ok.
  - **Bemerkung des Autors:** In dieser Hinsicht soll in Zukunft weiter gearbeitet werden.
- Wie gut erkennt man die einzelnen Einträge? Waren sie zu groß oder waren sie zu klein?
  - **Antwort der Expertin:** Durch die Möglichkeit, sehr weit reinzoomen zu können, sind die einzelnen Items gut erkennbar, allerdings verliert man den Überblick (Overview). Für mein Gefühl kann man zu weit einzoomen – in der großen Ansicht (ganz weit eingezoomt) sind die Items zu groß. Im ausgezoomten Zustand sind sie zu klein, das lässt sich klarer Weise nicht verändern, deshalb ist die Zoomfunktion sehr wichtig, aber auch ein Highlighting des gewählten Elementes, um sicher zu sein, dass auch wirklich das Item ausgewählt ist, was ich auswählen wollte.
- Waren die Zahlen bei den Koordinaten gut erkennbar?
  - **Antwort der Expertin:** Ich nehme mal an, dass es sich um die Achsenbeschriftung handelt: Wenn sie zu sehen waren, dann ja. Wobei ich sie nicht als Rahmen anzeigen würde, sondern ausgefüllt.
- Haben Sie bei der Installation oder beim Starten der Applikation Probleme gehabt?
  - **Antwort der Expertin:** Nein, überhaupt nicht.
- Haben Sie irgendwelche weiteren Bemerkungen zur Applikation?
  - **Antwort der Expertin:** Nein, ich denke, ich habe alles, was mir derzeit einfällt, schon aufgeschrieben.

Einige Bemerkungen, wie z.B. ein Overlay beim Tippen auf die Einträge, betreffen nicht direkt den AndroidPrefuse, sondern die Streudiagrammimplementierung. Die ursprüngliche Streudiagrammimplementierung in Desktop-prefuse war ein Tutorial für prefuse und

bewusst vom Autor einfach gehalten. Deswegen ist diese Implementierung auch nicht zu 100% für einen Enduser geeignet. Nicht desto trotz sind alle Bemerkungen, die die Expertin gemacht hat, für AndroidPrefuse und die Streudiagrammimplementierung sehr nützlich. Die Verbesserungen werden aber nicht im Zuge dieser Arbeit gemacht, sondern werden für die Zukunft vermerkt.

Im Voraus wurden einige Bemerkungen erwartet. Da in der Visualisierung eine Legende fehlt, wurde die Frage erwartet, welche Merkmale in welcher Form abgebildet sind, also welches Merkmal als x-Achse und welches als y-Achse dient. Ebenfalls wurde die Frage erwartet, was die Sterne und die Kreise darstellen. Da aber der Expertin erzählt wurde, dass es sich um Testdaten handelt, sind diese Fragen nicht gestellt worden.

4.8 Zusammenfassung

Nach einer Analyse der Grafikbibliotheken in den zwei Systemen, java Desktop und Android, und nach Analyse von prefuse, standen zwei Möglichkeiten für die Portierung von prefuse nach Android zur Verfügung: 1. Neuprogrammierung von prefuse in Android und 2. Übernahme der quelloffenen java-Desktop-Grafikbibliotheken AWT und Swing in Android. Wegen des viel geringeren Aufwands wurde die zweite Variante für die Portierung gewählt. Da die grafischen Funktionen bei den beiden System unterschiedlich sind, musste eine Brücke zwischen den beiden Systemen aufgebaut werden. Dies wurde mit der Klasse Android2Graphics erreicht. Beim Aufrufen der AWT- und Swing-Funktionen arbeitet diese Klasse und ruft die entsprechenden Android-Funktionen auf. Das Endresultat der Portierung wurde AndroidPrefuse genannt. Bei der Portierung wurden auch einige Anpassungen an AndroidPrefuse vorgenommen. Die Controllers, die für die Benutzerinteraktionen zuständig sind, wurden z.B. fast vollständig umgebaut. Um die Portierung zu demonstrieren, wurde ein Streudiagramm mit AndroidPrefuse implementiert. Bei der Portierung und Implementierung des Streudiagramms gab es einige Probleme (siehe Kapitel 4.5). Die Schwierigkeit, deren Lösung am aufwendigsten war, war das Performance-Problem. Um dieses Problem zu lösen, wurde die Möglichkeit in AndroidPrefuse eingebaut, die Einträge (bei Streudiagrammen die Punkte) in mehreren unabhängigen Threads zu verarbeiten. Damit wurde die Performance der Streudiagramm-Implementierung verdoppelt.

Mit der erfolgreichen Portierung von prefuse nach Android wurde die zweite Forschungsfrage komplett beantwortet. Konkret geht es um die zweite Teilfrage: „Sind die Design-Patterns, die in prefuse vorkommen, auch für Android geeignet?“. Also wurde in der Praxis bewiesen, dass die in prefuse verwendeten Design-Patterns sich auch bei Android anwenden lassen.

In Unterkapitel 4.6 wurde die dritte Forschungsfrage beantwortet, indem die vielen Vorteile von AndroidPrefuse gegenüber den wenig vorhandenen Visualisierungsbibliotheken in Android aufgelistet werden.

5 Beantwortung der Forschungsfragen

In diesem Kapitel wird noch einmal zusammenfassend auf die Forschungsfragen eingegangen.

1. Welche Visialisierungsbibliotheken gibt es bereits unter Android?

Diese Frage wurde komplett in Kapitel 2 beantwortet. Die bekanntesten Visualisierungsbibliotheken in Android sind: AchartEngine, Androidplot, GraphView, HoloGraphLibrary und charts4j. In Tabelle 1 sind sie zusammenfassend dargestellt, gemeinsam mit der Anzahl an Visualisierungen, die sie unterstützen, ob es möglich ist, neue Visualisierungen mit ihnen zu erstellen, und wie aufwendig dies ist. In dieser Tabelle ist auch prefuse aufgelistet, um einen direkten Vergleich von prefuse mit anderen Bibliotheken ziehen zu können. Die oben genannten Bibliotheken sind fast die einzigen in Android. Daraus lässt sich schließen, dass die Liste der Visualisierungsbibliotheken nicht besonders umfangreich ist. Dazu kommen auch die überschaubaren Visualisierungen, die sie anbieten, und die stark eingeschränkten Möglichkeiten, neue Visualisierungen zu erstellen.

2. Welche Software-Design-Patterns kommen bei (Desktop) prefuse vor? Sind sie auch für Android geeignet?

Bei der Entwicklung von prefuse haben seine Autoren genau überlegt, was sie mit dem Tool erreichen wollen. Heer et al [22] schrieben, dass sie ein Framework erstellen wollten, mit dem man neue Visualisierungen leicht und mit wenig Aufwand erstellen kann. Um dies zu erreichen, haben die Autoren bei der Entwicklung so viele Software-Design-Patterns wie möglich verwendet. In Kapitel 3.6.2.1 werden darin verwendete Design-Patterns aufgelistet und erklärt, ob sie sich für Android eignen würden. Dass diese Patterns sich für Android eignen, wurde theoretisch in Kapitel 3.6.2.1 dargelegt. Da die von prefuse verwendeten Design-Patterns keine umgebungsspezifischen Merkmale besitzen, können sie in jeder Umgebung mit einer objektorientierten Programmiersprache (inklusive Android), eingesetzt werden. Zusätzlich wurde diese Frage mit der erfolgreichen Portierung in der Praxis beantwortet. Die Tatsache, dass bei der Portierung keines der Design-Patterns verändert bzw. weggelassen wurde, spricht dafür, dass diese Frage durch die Portierung beantwortet wurde.

3. Welche Vorteile würde ein Visualisierungs-Framework wie prefuse für Android bieten?

Da Desktop-Rechner eine längere Geschichte als mobile Geräte haben, ist die Liste der Desktop-Applikationen für Visualisierungen dementsprechend länger und reicher. Mit dem zunehmenden Verkaufswachstum mobiler Geräte und andererseits fallenden Verkaufszahlen der Desktop-Rechner (siehe Abbildung 1) stellt sich die Frage, ob Desktop-Rechner überhaupt eine Zukunft haben. Einige Wissenschaftler/innen prognostizieren den „Tod“ des Desktop-Rechners. Auf der Webseite „Death of the Desktop - Envisioning

Visualization without Desktop Computing“ [43] publizieren Forscher/innen ihre wissenschaftlichen Arbeiten darüber, wie die Visualisierung in Zukunft ohne dem Desktop-Rechner werden könnte. Aus dieser Tatsachen könnte man schließen, dass es von Vorteil wäre, allgemein Desktop-Applikationen, soweit im mobilen Bereich nicht vorhanden, auf mobile Geräte zu portieren und anzupassen.

In Kapitel 2 wurden die vorhandenen Visualisierungsbibliotheken und deren Möglichkeiten vorgestellt. Es wurde festgestellt, dass sich diese nicht so gut für die Erstellung neuer Visualisierungen eignen. Außerdem ist die Anzahl dieser Bibliotheken in Android im Vergleich zu den Desktop-Applikationen sehr gering. prefuse und das, was prefuse anbietet, wurde in Kapitel 3.6 vorgestellt. prefuse ist sogar für die Desktop-Umgebung einzigartig, und fast kein anderes der vielen Visualisierungstools im Desktop-Bereich bietet vergleichbare Funktionen: die große Flexibilität und die Option, neue Visualisierungen leicht und mit wenig Aufwand zu erstellen – eine Möglichkeit, die die vorhandenen Visialisierungsbibliotheken in Android so gut wie gar nicht besitzen. Dies ist auch der größte Vorteil, den ein Visualisierungs-Framework wie prefuse in Android darstellen würde. Dazu kommt die Möglichkeit, vorhandene prefuse-Visualisierungen leicht nach Android zu portieren.

6 Ausblick

Der Fokus dieser Arbeit lag darauf, das Grundgerüst von prefuse nach Android zu portieren. Um dies zu demonstrieren, wurde ein funktionierendes Beispiel (Streudiagramm) erfolgreich portiert. Damit ist die vollständige Portierung von prefuse jedoch noch nicht abgeschlossen. Im Folgenden werden einige Arbeiten aufgelistet, die zur Vervollständigung der Portierung führen können.

Weitere Arbeiten in AndroidPrefuse

In Kapitel 4.2 wurde beschrieben, welche Teile nicht übernommen wurden und welche angepasst werden müssen.

Die Funktionen wie das Anzeigen, Bearbeiten bzw. Rendern von Bildern müssen in der Klasse Android2Graphics neu implementiert werden.

Viele UI-Komponenten aus dem Paket *prefuse.util.ui* wie JRangeSlider, JPrefuseTable, JForcePanel, JFastLabel, JsearchPanel, etc. müssen komplett neu implementiert werden.

Einige Controllers wie FocusControl, RotationControl, etc. und die entsprechenden EventHandlers, die nicht im implementierten Streudiagramm-Beispiel verwendet wurden, müssen ebenfalls neu implementiert werden. Da sich die Benutzerinteraktion in Android stark von denen auf dem Desktop unterscheidet, müssten höchstwahrscheinlich komplett andere Controllers implementiert werden, z.B. ein Controller und ein EventHandler für den Zwei-Finger-Wisch.

In Kapitel 4.5.2 wurden die Performance-Probleme dargelegt. Dafür wurde eine Teillösung gefunden. Deswegen wäre es angebracht, in Zukunft in dieser Richtung weiterzuarbeiten. Auch der Test mit der Expertin (siehe Kapitel 4.7) gibt Hinweise, dass in dieser Richtung Verbesserung notwendig ist. Einer der Ansätze für die Performance-Verbesserung wäre, die Verwendung der Klasse *java.awt.geom.GeneralPath* komplett durch die Android-Klasse *android.graphics.Path* zu ersetzen. Laut den Untersuchungen und Tests, die in Kapitel 4.5.2 beschrieben wurden, könnte diese Änderung ca. 9,5% Verbesserung in der Performance bringen. Weitere Analysen und Ermittlung von Teilen, die Performance-Verbesserung bringen können, wären für die Zukunft wünschenswert.

Implementierung / Portierung von prefuse-Demos

Um die vollständige Portierung von prefuse zu demonstrieren, müssten die mitgelieferten prefuse-Beispiele portiert werden. Diese unterscheiden sich sehr stark voneinander und jedes einzelne benötigt einiges an Aufwand und dementsprechend vielleicht Änderungen auch in AndroidPrefuse.

Implementierung von externen Visualisierungen

Wie schon an vielen Stellen dieser Arbeit bemerkt wurde, ist prefuse für die Erstellung individueller Visualisierungen sehr beliebt. Deswegen wäre ein logischer Schritt, die vorhandenen externen Visualisierungen auch in AndroidPrefuse zu implementieren. Weiters wäre die Portierung des Frameworks TimeBench, das auf prefuse aufgebaut ist, von Vorteil. Somit wären die zeitspezifischen Visualisierungen auch in Android verfügbar.

Vorgeschlagene Erweiterungen von der Expertin

In Kapitel 4.7 hat die Expertin eine Reihe von Verbesserungsvorschläge aufgelistet. Diese Vorschläge sind gute und sinnvolle Erweiterungen für AndroidPrefuse. Folgende Vorschläge/Bemerkungen wurden von der Expertin gegeben:

- Performance-Verbesserung beim Zoom- und Verschieb-Aktion

- Achsenbeschriftung auch beim Hineinzoomen anzeigen

- Anstatt einer Android Toast-Nachricht für das Anzeigen der Detaildaten zu verwenden, ein Overlay, was wegeklickt werden muss, verwenden

- Highlighten von dem ausgewählten/getippten Eintrag, damit man weiß welches Element gerade ausgewählt wurde. Dies ist nützlich, wenn viele Einträge sehr nah an einander stehen.

- Auswahl von einem Eintrag mit dem Fingertipp besser organisieren, damit man in einem weit ausgezoomten Zustand, das gewünschte Eintrag besser treffen kann.

- Im in einem normalen Zustand (nicht ein- bzw. ausgezoomten Zustand) mit einem Doppeltipp den Bereich, wo der Doppeltipp betätigt wurde, einzoomen.

- Filter-Funktionen einbauen

7 Zusammenfassung

Visualisierungsbibliotheken auf Android gibt es bereits einige. Wegen fehlender wissenschaftlicher Publikationen zu diesem Thema konnte keine überzeugende wissenschaftliche Verbindung zu den Visualisierungsbibliotheken gefunden werden. Weiters konnten wegen der fehlenden Informationen keine Aussagen zur Architektur, Benutzbarkeit und Performance dieser Visualisierungsbibliotheken gemacht werden. Man müsste sie selbst einzeln installieren, ausprobieren und Benchmarks durchführen, wovon aus Zeitgründen abgesehen wurde. Anhand der Beschreibungen, die die Bibliothekanbieter auf den Webseiten angegeben haben, konnte festgestellt werden, dass sie aber sehr stark darauf konzentriert sind, eine bestimmte Anzahl an Diagrammen und ab und zu einige Benutzerinteraktionen anzubieten. Einen Visualisierungs-Framework im Stile von prefuse, mit dem Wissenschaftler/innen und Programmierer/innen aus dem InfoVis- Bereich ihre Visualisierungen in Android einfach und mit wenig Aufwand erstellen können, gibt es leider nicht. Schon mit diesen Fakten konnte die Forschungsfrage „Welche Vorteile würde ein Visualisierungs-Framework wie prefuse für Android bieten?“ teilweise beantwortet werden. Dazu kommen die speziellen visualisierungsspezifischen und auch allgemeinen Design-Patterns in prefuse, wie Referenz-Modell-, Data Column-, Cascaded Tables-, Strategy-, Renderer-Design-Pattern, usw., die ebenfalls Vorteile der Portierung von prefuse nach Android darstellen. Diese Design-Patterns tragen nicht nur zur Wartbarkeit und Erweiterbarkeit von prefuse bei, sondern haben auch eine große Rolle bei der Entwicklung von AndroidPrefuse getragen. Sie spielen weiterhin dieselbe Rolle auch in AndroidPrefuse: Somit ist es ein Framework, das Wissenschaftler/innen und Programmierer/innen aus dem Informationsvisualisierungsbereich hilft, mit wenig Aufwand neue Visualisierungen zu entwickeln. Dies wird durch die Aufteilung einer Visualisierung auf viele unabhängige Komponenten erreicht. Die Entwickler/innen neuer Visualisierungen können dann einen Großteil der vorhandenen Teilkomponenten wiederverwenden, was viel Aufwand erspart.

Durch die Design-Patterns und vor allem durch die Übernahme der Bibliotheken AWT und Swing wurde die Portierung mit relativ wenig Aufwand durchgeführt. Die Vorteile dieses Vorgangs, der für die Portierung ausgewählt wurde, sind einerseits der leichte Einstieg für erfahrene prefuse-Entwickler/innen in AndroidPrefuse, andererseits die einfache Portierung von bereits vorhandenen Visualisierungen in prefuse. Was auf der einen Seite ein Vorteil ist, stellt gleichzeitig auch einen Nachteil dar. Durch die Übernahme von AWT- und Swing-Klassen ist man gezwungen, die grafischen Komponenten von AndroidPrefuse durch diese Klassen zu implementieren. Also sind Android-Benutzer/innen zum Großteil gezwungen, in Swing und AWT zu programmieren. Deswegen müssen sich erfahrene Android Grafikentwickler/innen, die wenig bzw. keine Erfahrung mit der Programmierung mit AWT und Swing haben, zusätzlich Wissen über diese Bibliotheken aneignen.

Literaturverzeichnis

[1] D. A. G. Aguilar, R. Therón, and F. J. G. Peñalvo, “Understanding Educational Relationships in Moodle with ViMoodle.,” presented at the Advanced Learning Technologies, 2008. ICALT ’08. Eighth IEEE International Conference on, Santander, Cantabria, 2008, IEEE, pp. 954–956.

[2] R. Ahmed and J. Aule, “An Evaluation of the Framework Libgdx when Developing a Game Prototype for Android Devices.,” Degree Project in Computer Science, KTH - Royal Institute of Technology, Sweeden, 2011.

[3] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski, “Visualizing Time-Oriented Data – A Systematic View,” *Computers & Graphics*, vol. 31, no. 3, pp. 401–409, Jun. 2007.

[4] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of Time-Oriented Data*. Springer, 2011.

[5] B. B. Bederson, B. Shneiderman, and M. Wattenberg, “Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies,” *ACM Trans. Graph.*, vol. 21, no. 4, pp. 833–854, Oct. 2002.

[6] M. Bostock and J. Heer, “Protovis: A Graphical Toolkit for Visualization,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1121–1128, Nov. 2009.

[7] M. Bostock, V. Ogievetsky, and J. Heer, “D^3 data-driven documents,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 2301–2309, 2011.

[8] P. Brady, “Anatomy & Physiology of an Android,” presented at the Google I/O 2008, https://sites.google.com/site/io/anatomy--physiology-of-an-android, San Francisco, Kalifornien, 28-May-2008.

[9] S. Card, J. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[10] A. Charland and B. Leroux, “Mobile application development: web vs. native,” *Communications of the ACM*, vol. 54, no. 5, pp. 49–53, 2011.

[11] L. Chittaro, “Visualizing information on mobile devices,” *Computer*, vol. 39, no. 3, pp. 40–45, Mar. 2006.

[12] P. Christensson, “Tooltip,” *TechTerms.com*. [Online]. Available: http://techterms.com/definition/tooltip. [Accessed: 17-Apr-2015].

[13] J. A. Cottam and A. Lumsdaine, “ThisStar: Declarative visualization prototype,” in *IEEE Symposium on Information Visualization*, 2007.

[14] B. Cronkite-Ratcliff and V. Pande, “MSMExplorer: visualizing Markov state models for biomolecule folding simulations,” *Bioinformatics*, p. btt051, 2013.

[15] S. Delap, “Google’s Android SDK Bypasses Java ME in Favor of Java Lite and Apache Harmony,” *http://www.infoq.com/news/2007/11/android-java*, 11-Dec-2007. [Online]. Available: http://www.infoq.com/news/2007/11/android-java. [Accessed: 14-Apr-2015].

[16] J.-D. Fekete, “The InfoVis Toolkit,” in *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, Austin, TX, 2004, IEEE, pp. 167–174.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[18] M. Giereth and T. Ertl, “Design Patterns for Rapid Visualization Prototyping,” in *Information Visualisation, 2008. IV ’08. 12th International Conference*, London, 2008, IEEE, pp. 569–574.

[19] P. I. A. Godinho, B. S. Meiguins, A. S. Goncalves Meiguins, R. M. Casseb do Carmo, M. de Brito Garcia, L. H. Almeida, and R. Lourenco, “PRISMA - A Multidimensional Information Visualization Tool Using Multiple Coordinated Views,” in *Information Visualization, 2007. IV ’07. 11th International Conference*, Zurich, 2007, IEEE, pp. 23–32.
[20] Google, “Hardware Acceleration, Unsupported Drawing Operations,” *developer.android.com*. [Online]. Available: http://developer.android.com/guide/topics/graphics/hardware-accel.html#unsupported. [Accessed: 05-Apr-2015].
[21] J. Heer and M. Agrawala, “Software Design Patterns for Information Visualization,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 12, no. 5, pp. 853–860, Sep. 2006.
[22] J. Heer, S. K. Card, and J. A. Landay, “Prefuse: A Toolkit for Interactive Information Visualization,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2005, ACM, pp. 421–430.
[23] R. Hijon-Neira and J. A. Velazquez-Iturbide, “How to Improve Assessment of Learning and Performance through Interactive Visualization,” in *Advanced Learning Technologies, 2008. ICALT ’08. Eighth IEEE International Conference on*, Santander, Cantabria, 2008, IEEE, pp. 472–476.
[24] J. de Jesus Nascimento da Silva Junior, B. S. Meiguins, N. S. Carneiro, A. S. G. Meiguins, R. Y. da Silva Franco, and A. G. M. Soares, “PRISMA Mobile: An Information Visualization Tool for Tablets,” in *Information Visualisation (IV), 2012 16th International Conference on*, Montpellier, 2012, IEEE, pp. 182–187.
[25] P. O. Kristensson, O. Arnell, A. Björk, N. Dahlbäck, J. Pennerup, E. Prytz, J. Wikman, and N. Åström, “InfoTouch: An Explorative Multi-touch Visualization Interface for Tagged Photo Collections,” in *Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges*, New York, NY, USA, 2008, ACM, pp. 491–494.
[26] T. Lammarsch, W. Aigner, A. Bertone, S. Miksch, T. Turic, and J. Gartner, “A Comparison of Programming Platforms for Interactive Visualization in Web Browser Based Applications,” in *Information Visualisation, 2008. IV ’08. 12th International Conference*, London, 2008, IEEE, pp. 194–199.
[27] R. R. Moelker and W. E. Wijbrandi, “HTML5 data visualization capabilities of mobile devices,” *9th SC@ RUG 2011-2012*, p. 23, 2011.
[28] J. Nielsen, “Response Times: The 3 Important Limits,” 01-Jan-1993. [Online]. Available: http://www.nngroup.com/articles/response-times-3-important-limits/. [Accessed: 16-Apr-2015].
[29] D. Nolan and D. T. Lang, “Interactive and animated scalable vector graphics and R data displays,” *Journal of Statistical Software*, vol. 46, no. 1, pp. 1–88, 2012.
[30] C. Pruett, “Writing real time games for Android,” presented at the Google I/O 6, May-2009.
[31] A. Rind, “prefuse Tutorial: Scatter Plot,” 10-May-2009. [Online]. Available: http://www.ifs.tuwien.ac.at/~rind/w/doku.php/java/prefuse-scatterplot. [Accessed: 18-Aug-2015].
[32] A. Rind, T. Lammarsch, W. Aigner, B. Alsallakh, and S. Miksch, “TimeBench: A Data Model and Software Library for Visual Analytics of Time-Oriented Data,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 19, no. 12, pp. 2247–2256, Dec. 2013.
[33] B. Shneiderman, “Tree Visualization with Tree-maps: 2-d Space-filling Approach,” *ACM Trans. Graph.*, vol. 11, no. 1, pp. 92–99, Jan. 1992.

[34] Statista, “Prognose zum Absatz von Tablets, PCs und Smartphones weltweit von 2012 bis 2018,” 2015. [Online]. Available: http://de.statista.com/statistik/daten/studie/256337/umfrage/prognose-zum-weltweiten-absatz-von-tablets-pcs-und-smartphones/. [Accessed: 14-Apr-2015].
[35] Statista, “Prognose zu den Marktanteilen der Betriebssysteme am Absatz vom Smartphones weltweit in den Jahren 2014 und 2018,” 2015. [Online]. Available: http://de.statista.com/statistik/daten/studie/182363/umfrage/prognostizierte-marktanteile-bei-smartphone-betriebssystemen/. [Accessed: 14-Apr-2015].
[36] R. C. Team, *R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, 2012*. ISBN 3-900051-07-0, 2012.
[37] M. Tory and T. Möller, “Human factors in visualization research,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 10, no. 1, pp. 72–84, 2004.
[38] P. Weishapl, “TimeVis Visualizing Temporal Data using prefuse,” Bachelor Thesis, Vienna University of Technology, Vienna, 2007.
[39] “Apache Harmony - Use in Android SDK,” *Wikipedia*. [Online]. Available: http://en.wikipedia.org/wiki/Apache_Harmony#Use_in_Android_SDK. [Accessed: 19-Apr-2015].
[40] “Säulendiagramm,” *Wikipedia*. [Online]. Available: https://de.wikipedia.org/wiki/Säulendiagramm. [Accessed: 27-Sep-2015].
[41] “GNU Classpath.” [Online]. Available: http://www.gnu.org/software/classpath/faq/faq.html. [Accessed: 17-Apr-2015].
[42] “Antialiasing,” *Wikipedia*. [Online]. Available: http://de.wikipedia.org/wiki/Antialiasing_%28Computergrafik%29. [Accessed: 17-Apr-2015].
[43] “Death of the Desktop - Envisioning Visualization without Desktop Computing.” [Online]. Available: http://beyond.wallviz.dk/. [Accessed: 16-Oct-2015].