Visualisierung von mobilen Sensordaten

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Georg Bachmann

Institut für Wissensmanagement (IWM),

Technische Universität Graz

A-8010 Graz

Mai 2012

Diese Arbeit ist in deutscher Sprache verfasst.

Begutachterin: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Betreuerin: Dr. Viktoria Pammer-Schindler

Visualization of mobile sensor data

Master's Thesis

at

Graz University of Technology

submitted by

Georg Bachmann

Knowledge Management Institute (KMI),

Graz University of Technology

A-8010 Graz, Austria

May 2012

This thesis is written in german language

Evaluator: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Advisor: Dr. Viktoria Pammer-Schindler

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Danksagungen

Ich möchte an dieser Stelle vor allem meinen Eltern für ihre Unterstützung während meines gesamten Studiums danken. Vielen vielen lieben Dank Mama und Papa!

Des Weiteren würde ich diese Gelegenheit gerne nutzen meiner Freundin zu danken, die mir während des schriftlichen Teils dieser Arbeit so viel Geduld und Verständnis entgegengebracht hat! Danke Babsi!

Darüber hinaus will ich auch meinem guten Freund und Kollegen Stefan Edler danken, der mir während meiner Masterarbeit immer unterstützend zur Seite gestanden ist.

Zu guter Letzt danke ich Frau Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt und dem Know-Center, die mir die Arbeit an dieser Masterarbeit ermöglicht haben. Mein Dank gilt auch meiner Betreuerin Frau Dr. Viktoria Pammer-Schindler, die mir immer mit Tipps und Ratschlägen hilfreich zur Seite gestanden ist.

Georg Bachmann

Graz, Mai 2012

Abstract

When looking at the development of mobile devices in the past few years, it is obvious that smartphones and tablet computers are gaining more and more relevance. In Austria alone, smartphones constitute around one third of all mobile phones in use. Not only do smartphones get faster processors, more efficient graphic cards and more memory with each revision, but they also gain new sensors with publicly available APIs for developers. This makes smartphones suitable for the use as mobile sensing devices and interesting for researchers who want to capture data about human behavior (movement, communication, daily patterns, etc.) in real life. However, once all this data is captured, the question is how to efficiently visualize all those numbers. This visualization of mobile sensor data directly on smartphones and tablets is the main topic of this master thesis. As a first step an exact analysis of the data that needs to be visualized is done, as well as an overview of the main challenges that developers face when building mobile applications, namely limited hardware resources and specific user interaction paradigms. In a second step this thesis introduces well-known visualizations that are able display all kinds of different data. Before actually presenting the practical work that was done as part of this thesis, related work is described and compared to each other. This master thesis takes a look at the different approaches on how to effectively present data to better visualize it on a mobile device. The main part of this thesis describes the visualization framework that was developed to display mobile sensor-data in a performant and interactive way directly on mobile devices. This visualization framework was integrated with a sensing framework into a fully functional prototype (iPeeper) for capturing, displaying and synchronizing sensor data across multiple devices.

Kurzfassung

Betrachtet man die Entwicklung von mobilen Geräten der letzten Jahre, sieht man, dass Smartphones und Tablets immer mehr an Bedeutung gewinnen. Alleine in Österreich machen Smartphones bereits rund ein Drittel aller Mobiltelefone aus. Diese Geräte bringen aber nicht nur von Generation zu Generation schnellere Prozessoren, leistungsstärkere Grafikkarten und mehr Speicher, sondern auch immer mehr Sensoren die mittels APIs auslesbar sind. Das bietet Wissenschaftlern die Daten über das menschliche Verhalten (Bewegungen, Kommunikation, tägliche Abläufe, etc.) im echten Leben aufzeichnen wollen, eine sehr einfache Möglichkeit Benutzerdaten von einer möglichst breiten Zielgruppe zu erhalten. Nachdem diese Sensoren aber nun alle nur erdenklichen Informationen gesammelt haben, stellt sich die Frage nach einer passenden Visualisierung all dieser Zahlen.

Diese Masterarbeit beschäftigt sich mit genau dieser Visualisierung von mobilen Sensordaten direkt auf mobilen Endgeräten. In einem ersten Schritt wird eine genaue Analyse der Aufgabenstellung durchgeführt und auf die zu visualisierenden Sensordaten, die vorherrschenden Limitierungen von mobilen Geräten hinsichtlich von Hardware-Resourcen sowie auf die speziellen User-Interaktions Paradigmen auf mobilen Geräten eingegangen. Weiters werden in dieser Arbeit grundsätzliche Visualisierungen vorgestellt, die es ermöglichen sehr viele verschiedene Arten von Daten effizient darzustellen. Nach einer genaueren Beleuchtung und einem Vergleich von ähnlichen Arbeiten, beschreibt der Hauptteil dieser Masterarbeit die Umsetzung eines Visualisierungs-Frameworks, dass eine performante und interaktive Darstellung von mobilen Sensordaten direkt am Smartphone bzw. Tablet erlaubt. Dieses Visualisierungs-Framework wurde mit einem Sensing-Framework zu einem voll funktionsfähigen Prototypen namens iPeeper kombiniert, der Sensordaten aufzeichnet, darstellt, und über mehrere Geräte synchronisiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

"The purpose of computing is insight, not numbers" *[Hamming, 1987]*

1.1 Problemstellung

User-Context Aufzeichnung findet im Desktop Bereich schon seit einiger Zeit Anwendung, sei es im wissenschaftlichen als auch im privaten Bereich. Software Sensoren erkennen was der Benutzer am Computer tut, zeichnen auf welche Programme wie lange und wann offen sind und erkennen Zusammenhänge zwischen diesen Programmen. Das kann passieren es wenn ein Benutzer beispielsweise per Drag and Drop Dateien verschiebt, oder per Zwischenablage Daten aus einem Programm kopiert und in ein anderes einfügt [Rath, 2010] [Rath et al., 2008] [Abowd et al., 1999] [Borodin et al., 2007].

Smartphones werden immer erschwinglicher und der Marktanteil macht alleine in Österreich bereits ein Drittel des Handymarktes aus (Quelle: Austrian Internet Monitor (AIM), Stand drittes Quartal 2011[1]). Somit werden Smartphones auch für die User-Context Erfassung immer interessanter. Des Weiteren haben diese Geräte immer mehr Sensoren eingebaut haben, die Aufschlüsse über das Verhalten des Benutzers ermöglichen. Somit kann der Einsatz von mobilen Geräten zur Erfassung von User-Context Daten dabei helfen, dem Benutzer seine Gewohnheiten auch abseits des computergestützten Arbeitsplatzes aufzuzeigen [Eagle und Pentland, 2005].

Eine spannende Aufgabenstellung in diesem Zusammenhang ist die visuelle Aufbereitung von derartigen User-Context Daten direkt auf mobilen Geräten. Die Herausforderung

[1] http://www.integral.co.at/downloads/Internet/2011/07/AIM-Consumer_Pressetext_-_Q2_2011.pdf

besteht einerseits in der performanten visuellen Aufbereitung dieser Daten, die dem Benutzer auch eine Interaktion ermöglicht. Mobile Geräte verfügen über stark limitierte Ressourcen im Vergleich zu Desktop Systemen, sei es im Hinblick auf die Displaygröße, aber auch im Hinblick auf die Leistung der CPU bzw. GPU und der stark limitierte Speicher. Zum anderen liegt die Herausforderung darin, auf gängige mobile User-Interaktions Paradigmen der mobilen Plattform einzugehen und diese bestmöglich umzusetzen.

Unter Berücksichtigung dieser Herausforderungen sollte es dem Benutzer möglich sein, die aufgezeichneten Daten direkt am mobilen Endgeräte so darzustellen, dass sie performant und interaktiv betrachtet werden können [Boud, 1985] [Shneiderman, 2008] [Borodin et al., 2007].

1.2 Zielbeschreibung

Ziel dieser Masterarbeit ist es ein Visualisierungs-Framework für mobile Geräte zu entwickeln, das es dem Benutzer ermöglicht interaktiv, performant und unter Berücksichtigung geltender UI-Paradigmen, aufgezeichnete mobile Sensordaten direkt am mobilen Endgerät zu visualisieren. Dabei muss natürlich auch auf die Probleme Rücksicht genommen werden, die mobile Geräte in Hinblick auf Leistung mit sich bringen. Auch die Interaktions-Patterns von mobilen touch-basierten User-Interfaces werden berücksichtigt und ausgenutzt um die Benützung möglichst einfach und intuitiv zu gestallten.

1.3 Aufbau der Arbeit

Bevor genauer auf die Lösung eingegangen werden kann, die im Zuge dieser Masterarbeit erarbeitet wurde, wird zuerst in Kapitel 2 die Problemstellung im Detail analysiert. Das umfasst eine ausführliche Beleuchtung der mobilen Sensordaten die visualisiert werden können, aber auch die genauere Erläuterung der somit entstehenden Herausforderungen im Bezug auf das mobile Umfeld der Arbeit. Kapitel 3 geht dann darauf ein welche Möglichkeiten es im Generellen gibt, mobile Sensordaten effizient darzustellen. In Kapitel 4 werden Arbeiten vorgestellt die im Zusammenhang mit dieser Masterarbeit relevant sind, die sich also mit dem performanten Visualisieren von großen Datenmengen beschäftigen. Im Anschluss wird in Kapitel 5 auf die Implementierung des praktischen Teils dieser Masterarbeit eingegangen. Zuerst gibt das Kapitel einen generellen Überblick über das Visualisierungs-Framework, das entwickelt wurde und geht dann auf die einzelnen Visualisierungen im De-

tail ein, die Teil dieses Frameworks sind. Weiters werden auch Lösungen vorgestellt, die es dem Benutzer ermöglichen, aufgezeichnete Sensordaten interaktiv zu betrachten. Als Teil dieser Masterarbeit ist neben dem Visualisierungs-Framework auch ein Prototyp namens iPeeper entstanden, der in Kapitel 6 genauer beschrieben wird. iPeeper ist in Kooperation mit [Edler, 2012] entstanden, der sich mit dem Aufzeichnen und Verarbeiten von mobilen Sensordaten beschäftigt. Mögliche Anwendungsszenarien für diesen Prototypen werden in Kapitel 7 beschrieben. Abschließend wird in Kapitel 8 das Ergebnis dieser Arbeit diskutiert, sowie mögliche Anknüpfungspunkte für zukünftige Arbeiten beschrieben.

Kapitel 2

Problemstellung

"Auch wenn neuere Untersuchungen zeigen, dass jeder Mensch einen bevorzugten Eingangskanal hat und dies nicht in jedem Fall der visuelle ist, bleibt die Tatsache unbestritten, dass der Mensch ein 'Augentier' ist. Die meisten Menschen sind (zumindest auch) 'visuelle Typen'" *[Seifert, 2003, S. 11]*

Ziel dieser Masterarbeit war es, ein Visualisierungs-Framework zu entwerfen, mit dessen Hilfe es möglich ist, mobile Sensordaten interaktiv und möglichst performant auf mobilen Geräte anzuzeigen. In diesem Kapitel wird näher beschrieben welche Daten visualisiert werden sollen und welche Herausforderungen die interaktive Informationsvisualisierung auf mobilen Endgeräten darstellt.

2.1 Welche Daten sollen visualisiert werden

Um zu wissen welche Visualisierungen man zum Darstellen von mobilen Sensordaten benötigt, macht es Sinn, sich zuerst zu überlegen, welche Daten man überhaupt visualisieren will. Dieses Unterkapitel behandelt die Sensoren und deren Werte, die iOS Geräte ohne extra Berechnungen zur Verfügung stellen. Auf eventuelle Schlüsse, die man durch eine genauere Analyse dieser Werte kommen kann, wird in [Edler, 2012] genauer eingegangen.

2.1.1 GPS (Global Positioning System)

Um es mobilen Applikationen zu erlauben geographisch bezogene Informationen anzuzeigen, haben sehr viele gängige Smartphones einen GPS Sensor eingebaut. Ein GPS Sensor liefert die geografische Länge, Breite und Höhe eines Punktes. Weiters ist zu jedem Wert

auch der genaue Messzeitpunkt bekannt. Wenn man nun mehrere Punkte miteinander in Verbindung setzt, kann man auch die Geschwindigkeit und Richtung einer eventuellen Bewegung berechnen.

Somit können folgende Daten aufgrund von GPS Messungen entstehen:

geografische Länge, Breite	float [in Grad]
geografische Höhe	float [in m]
Zeitpunkt der Messung	date
Geschwindigkeit	float [in km/h]
Richtung	float [in Grad]

2.1.2 Beschleunigungssensor

Damit ein Smartphone beispielsweise auf eine Rotation des Gerätes von Hochformat auf Querformat das User-Interface entsprechend anpassen kann, haben viele moderne Mobilgeräte einen Beschleunigungssensor eingebaut. Ein Beschleunigungssensor misst jegliche Beschleunigung die auf ihn einwirkt. Somit liefert er Beschleunigungswerte an der x, y, und z Achse des Gerätes.

Daraus ergeben sich folgende verfügbare Werte:

Beschleunigung um die X-Achse	float [in g]
Beschleunigung um die Y-Achse	float [in g]
Beschleunigung um die Z-Achse	float [in g]
Zeitpunkt der Messung	date

2.1.3 Mikrofon

Da ein Telefon ursprünglich hauptsächlich zur Übertragung von Gesprächen gedacht war, haben alle Smartphones, aber auch viele heutzutage gängige Tablet Computer, ein Mikrofon eingebaut. Dieses dient dazu Gespräche aufzuzeichnen und in digitale Informationen umzuwandeln. [Lu et al., 2009]

Da so aufgezeichnete Gespräche aber sehr sensible Informationen beinhalten können, wurden im Rahmen dieser Masterarbeit, die Mikrofondaten auf die Umgebungslautstärke reduziert. Verfügbar wäre aber natürlich auch das Gesprochene selbst.

Gespräch bzw. Geräusch	Audiodatei (wurde für diese Masterarbeit nicht gespeichert)
Lautstärke	float [in db]
Zeitpunkt bzw. Zeitraum der Messung	date

2.1.4 Umgebungslichtsensor

Moderne Smartphones verfügen über einen Umgebungslichtsensor. Dieser dient meist dazu, die Helligkeit des Displays an das Umgebungslicht anzupassen. So wird es zum Beispiel in dunklen Umgebungen auch dunkler und in hellen Umgebungen noch heller um genug Kontrast zu gewährleisten. Zum anderen dient dieser Sensor dazu zu erkennen ob das Telefon ans Ohr gehalten wird, um in diesem Fall das Display und vor allem die Touch-Steuerung zu deaktivieren, um keine unabsichtlichen Inputs mit dem Ohr zu generieren.

Helligkeit	float [von 0 (kein Licht) bis 1 (maximale Helligkeit)]
Zeitpunkt der Messung	date

2.1.5 Kamera

Da es heutzutage so gut wie kein Mobiletelefon mehr ohne Kamera gibt und ein Bild auch bekanntlich mehr als tausend Worte (im Fall der mobilen Context Erkennung wohl eher “Werte") sagt, ist die Kamera ein sehr attraktiver Sensor. Und wenn ein einzelnes Bild bereits mehr als tausend Worte sagt, ist das Abspielen von mehreren Bildern hintereinander wohl mit einem Roman gleichzusetzen.

Von Microsoft gibt es zu diesem Thema bereits einen spannenden Forschungs-Prototypen namens SenseCam[1]. Die SenseCam zeichnet in einem definierten Intervall Bilder (ebenso andere Sensordaten wie Temperatur und Beschleunigung) auf und fügt diese anschließend am Computer mit entsprechender Software zu einem Zeitraffervideo zusammen [Hodges et al., 2011] [Wood et al., 2004].

Daten, die eine Kamera für die mobile Context-Erkennung liefert, sind folgende:

Bild	Bilddatei [komprimiertes JPEG]
Zeitpunkt des Bildes	date

[1] http://research.microsoft.com/en-us/um/cambridge/projects/sensecam/ (zuletzt besucht am 16. März 2012)

2.1.6 Nachrichten

Smartphones können Nachrichten aller Art empfangen. Sei es eine Email, eine SMS oder einfach nur ein RSS Nachrichtenbeitrag. Allen Nachrichten, die in dieser Masterarbeit berücksichtigt wurden, ist gemeinsam, dass sie Text beinhalten und von einer identifizierbaren Quelle stammen. Diese Quelle kann der Absender sein, aber auch der Autor, der für eine Website einen Beitrag geschrieben hat.

Für die Visualisierungen, die im Zuge dieser Masterarbeit entstanden sind, werden folgende Daten berücksichtigt:

Text	string
Zeitpunkt der Nachricht	date
Quelle	string

2.1.7 Bluetooth

Schon seit langem verfügen viele Mobiltelefone über Bluetooth Chips, um mit anderen Geräten drahtlos Daten austauschen zu können. Ein solcher Bluetooth Chip kann aber auch dafür verwendet werden andere sich in der Nähe befindliche mobile Geräte zu detektieren, die auch Bluetooth aktiviert haben. Eine spannende Arbeit, die sich mit diesem Thema beschäftigt ist [Rudström et al., 2005].

Für den User-Context spannende Daten sind in der folgenden Tabelle abgebildet:

Gerätename	string
Zeitpunkt der Begegnung	date
Dauer der Begegnung	timespan [in Sekunden]

2.1.8 Kompass

Neben der Möglichkeit die Bewegungsrichtung eines Benutzers anhand der Differenz zweier Geo-Positionspunkte zu ermitteln, haben immer mehr Smartphones einen elektronischen Kompass eingebaut. Ein solcher Kompass ermöglicht die Bestimmung der Blickrichtung des Benutzers, auch wenn das mobile Geräte stationär an einem Ort bleibt. Vor allem beim Analysieren von Bildern kann diese Information in Kombination mit den genauen Koordinaten spannend sein, um den betrachteten Bildausschnitt besser interpretieren zu können.

Vom Kompass gelieferte relevante Daten sind somit:

Richtung	float [in Grad]
Zeitpunkt der Messung	date

2.1.9 Externe Sensoren

Des Weiteren ist es möglich externe Sensoren, wie zum Beispiel Pulssensoren oder Blutdruckmesser, an ein Mobiltelefon anzuschließen. Die Möglichkeiten die sich daraus ergeben, würden diese Liste noch stark erweitern. Für diese Masterarbeit wurden jedoch nur die in iOS Geräten eingebauten internen Sensoren genauer betrachtet. Eine Erweiterung auf externe Sensoren sollte aber bei entsprechender Aufbereitung der zur Verfügung gestellten Daten, kein Problem sein. So könnten beispielsweise Pulsdaten, ähnlich wie die von einem GPS-Sensor aufgezeichneten Höhenverlaufslinien, leicht mit dem bestehenden Visualisierungs-Framework dargestellt werden.

2.2 Herausforderungen im mobilen Umfeld

Als Plattform für die Implementierung des praktischen Teils dieser Masterarbeit wurde das iPhone bzw. das iPad gewählt. Die Herausforderungen, die man bei der Implementierung von grafiklastigen Applikationen auf mobilen Geräten hat, gelten aber prinzipiell für alle gängigen mobilen Plattformen.

2.2.1 Hardware Ressourcen

Die Hardware von heutigen Smartphones ist der eines Desktop Rechners noch stark unterlegen. Vor allem gilt das für die CPU Leistung und den eingebauten RAM Speicher. Gängige high-end Geräte verfügen derzeit über rund 1GHZ CPU Geschwindigkeit (manche bereits mit dual-core, andere noch single-core Prozessoren) und 256 - 512MB RAM Speicher. Vor allem bei sehr grafiklastigen Anwendungen kann man hier sehr schnell an Grenzen stoßen und muss viele Dinge anders lösen, als man sie auf einem Desktopsystem gelöst hätte.

2.2.2 Speicher

Eines der Hauptprobleme beim Darstellen von großen Grafiken ist der Speicherverbrauch von Visualisierungen. Um einen Pixel auf einem Gerät darzustellen, benötigt man mindestens 4 Byte Speicher. Dieser Speicherbedarf setzt sich folgendermaßen zusammen:

RGBA8888	
1 Byte	Rotanteil
1 Byte	Grünanteil
1 Byte	Blauanteil
1 Byte	Alpha (Transparenz) Anteil
4 Byte	Gesamtspeicher pro Pixel

Das Display des iPhones ist derzeit 640 x 960 Pixel groß. Das bedeutet das man 614400 Pixel zu füllen hat, was somit einen Mindest-Speicherverbrauch von 2457600 Byte (rund 2.3MB) zur Folge hat. Wenn man allerdings davon ausgeht, dass eine Visualisierung von sehr vielen Datenpunkten doch um einiges größer als eine Bildschirmgröße des iPhones wird, steigt der Speicherbedarf dementsprechend an.

Bei Testläufen mit dem in dieser Masterarbeit entstanden Prototypen iPeeper (siehe Kapitel 6), wurden bei einer Aufzeichnung einer Autofahrt mit rund 45km, über 3000 Höheninformationswerte aufgezeichnet. Wenn man diesen Höhenverlauf visualisieren will und pro Punkt einen Abstand von 20 Pixeln einhält, um eine bessere Lesbarkeit zu gewährleisten, würde man auf eine Visualisierung mit 60.000x960 Pixeln kommen, was rund 55MB Speicher benötigen würde, wenn man die Grafik vorgerendert im Speicher halten würde.

Um diesem Problem entgegenzuwirken, setzen die in dieser Masterarbeit implementierten Visualisierungen darauf, immer nur den sichtbaren Teil der Visualisierung zu rendern. Das hat zwar ein wenig Performance-Einbußen zur Folge, nimmt aber wesentlich weniger Speicher in Anspruch.

2.2.3 Touchscreens

Im Gegensatz zu einem konventionellen Betriebssystem das mittels Tastatur und Maus gesteuert wird, setzen touch-basierte Betriebssysteme meistens nur auf den User-Input mittels Finger oder Stylus. Der größte Unterschied zur Eingabe mittels Maus ist die Genauigkeit des Klicks. Die Fläche einer Fingerspitze ist wesentlich größer als die Spitze eines herkömmlichen Mauszeigers und somit ist es für den Benutzer um einiges schwieriger, genau eine exakte Stelle am Display anzuklicken. Für das Entwerfen von Applikationen, die für solche touch-basierten Geräte ausgelegt sind, ist es somit wichtig, das User-Interface entsprechend zu gestalten. Laut den Human Interface Guidelines von Apple [Apple, 2012e, S. 13] sollte eine solche klickbare Fläche mindestens 44 x 44 Punkte groß sein.

2.2.4 Bildschirmgröße

Heutige Smartphones besitzen im Durchschnitt eine Displaydiagonale von rund 3 - 4 Zoll, was rund 10 Zentimetern entspricht. Laut einer Statistik von w3schools[2] verfügen über 85% der im Web aktiven Desktop Systeme über eine Bildschirmauflösung von mehr als 1024x768 Pixeln. Das entspricht einer Bildschirmdiagonale von $\sqrt{1024^2px + 768^2px}$ = 1280px. Die durchschnittliche Auflösung von LCD Bildschirmen (von 72dpi bis 96dpi) ergibt im Schnitt 84dpi. Wenn man diese 84 dpi mit der Bildschirmdiagonale von 1280px verrechnet, ergibt das eine Bildschirmdiagonale von rund $\frac{1280px}{84dpi}$ = 15 Inch was rund 38 Zentimetern entspricht.

Somit kann man sehen, dass der Platz auf einem Smartphone vergleichsweise sehr stark limitiert ist. Deshalb ist es wichtig diesen Platz sehr effizient zu nutzen. Nicht wichtige Informationen sollten nur dann angezeigt werden, wenn sie für den Benutzer relevant werden. Gleiches gilt für UI-Elemente zur Steuerung der Applikation. Selten genutzte Funktionen sollten nicht viel Platz in Anspruch nehmen und optimalerweise erst dann erscheinen, wenn der Benutzer sie auch wirklich benötigt. Umso weniger Platz effektiv für die relevanten Informationen zur Verfügung steht, umso schwieriger wird es für den Benutzer diese Information auch wirklich zu verwerten [Maniar et al., 2008] [Apple, 2012e] [Google, 2012].

[2] http://www.w3schools.com/browsers/browsers_display.asp (Stand: 6. April 2012)

Kapitel 3

Grundlagen interaktiver Informationsvisualisierung

"Visualisieren bezeichnet die Tätigkeit, einen bislang im Zeichensystem der Wortsprache ausgedrückten Inhalt entweder durch bildsprachliche Zeichen zu ergänzen, oder aber ihn ganz in die Bildsprache zu übersetzen." *Stary Joachim*

Umso größer und komplexer Datensätze werden, desto komplizierter wird es, diese Daten effizient analysieren und vergleichen zu können. Für uns Menschen ist es unter Umständen schwerer Schlüsse zu ziehen, wenn wir nur lange Listen von Zahlen vor uns haben. Aus diesem Grund bedienen wir uns oft der Unterstützung von visuellen Hilfsmitteln, um es für uns einfacher zu machen, diese Flut an Daten effizient verarbeiten zu können. Wie Abbildung 3.1 zeigt, sieht sich die Informationsvisualisierung als Schnittstelle zwischen komplexen Datensätzen und dem Menschen [Kerren et al., 2008] [Borodin et al., 2007] [Shneiderman, 2008].

In vielen wissenschaftlichen Arbeiten findet man neuartige Formen von Visualisierungen. Als Beispiel sei hier das Affective Diary (Kapitel 4.4.1) oder MobiVis (Kapitel 4.4.2) genannt. Dabei wird nicht auf Standardvisualisierungen, wie Liniendiagramme oder Pie Charts zurückgegriffen, sondern es werden eigene, maßgeschneiderte Methoden entwickelt, um Informationen abzubilden. Das hat den Vorteil einer genau auf die darzustellenden Daten zugeschnittenen Visualisierungs-Lösung. Somit kann man sehr viel Information auf wenig Fläche unterbringen und vor allem die komplette Information in nur einer Visualisierung

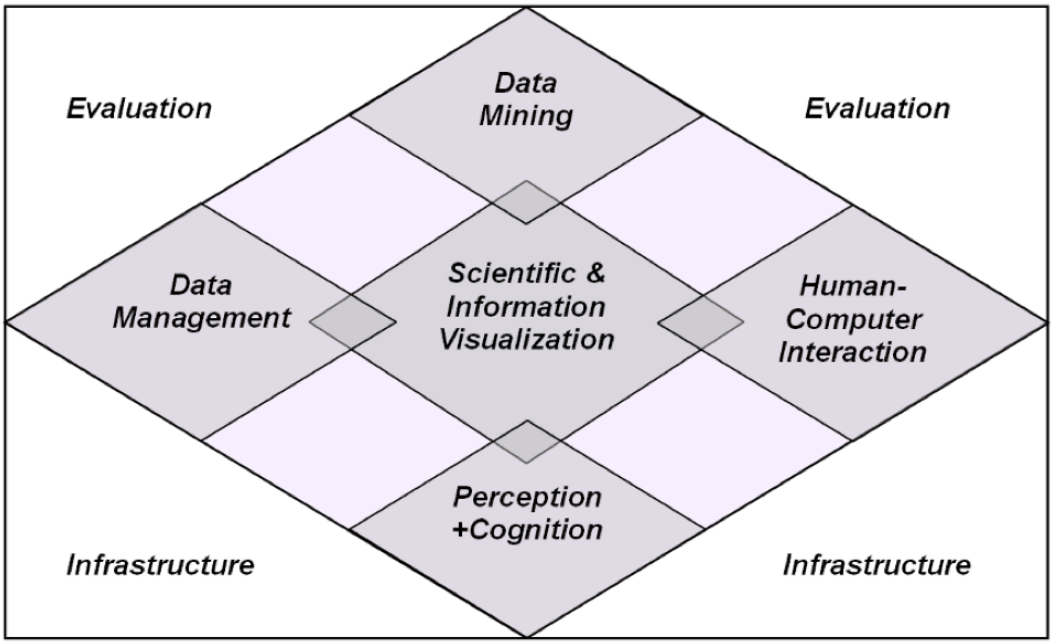

Abbildung 3.1: Visualisierungen als Schnittstelle zwischen komplexen Datensätzen und dem Menschen. Quelle: [Kerren et al., 2008, S. 159]

darstellen. Der Benutzer muss allerdings immer von neuem lernen, was die einzelnen Farben, Symbole, Formen, etc. zu bedeuten haben.

Aus diesem Grund, und aus dem, dass solche Visualisierungen meist sehr spezifisch nur einen Anwendungsfall abdecken, habe ich mich bei der Implementierung des VisualizationKit-Frameworks, das im praktischen Teil dieser Masterarbeit entwickelt wurde, für herkömmliche und generischere Visualisierungs-Methoden als Basis entschieden.

Da viele der Ansätze in solchen nicht "konventionellen" Visualisierungsformen, wie beispielsweise dem Affective Diary, allerdings sehr nützlich sind, sind auch Teile davon in das Visualisierungs-Framework, dass im Zuge dieser Masterarbeit entstanden ist, eingeflossen. So ist zum Beispiel das Kodieren von Informationen mit speziellen Farben ein sehr einfacher und platzsparender Weg um zusätzliche Informationen effizient darzustellen.

Da somit die Frage nach dem "Wie sollen die Daten visualisiert werden?" geklärt ist, ist nun die nächste Frage, was in den einzelnen Sensordaten einheitlich ist. Da jeder Sensor absolut verschiedene Daten liefern kann, war nur der Zeitpunkt der Messungen einheitlich, was somit die Verwendung eines Zeitstrahls naheliegen ließ. Was aber vor allem Dank dieser Zeitpunkte möglich ist, ist die Synchronisation zwischen den Visualisierungen. Mehr dazu können Sie in Kapitel 5.4 bzw. Kapitel 5.5 lesen.

3.1 Gängige User-Interface Metaphern für touch-basierte mobile Geräte

Um es dem Anwender möglichst einfach zu machen sich in einer Applikation zurecht zu finden, ist es wichtig auf gängige User-Interface Metaphern für die entsprechende Plattform zurückzugreifen. Für die beiden großen mobilen Betriebssysteme, iOS und Android, stellt Apple bwz. Human Interface Guidelines zur Verfügung, an die sich Entwickler halten sollten, um für Anwender möglichst konsistente Applikationen zu entwerfen [Apple, 2012e] [Google, 2012].

Gesten die für die meisten touch-basierten mobile Geräte gängig sind, sind folgende:

- **Touch**

  Ein Touch ist mit einem Doppelklick in einem Desktop System, dass mit einer Maus gesteuert wird, gleich zu setzen. Der Benutzer tappt einmal kurz auf den Bildschirm und hebt seinen Finger gleich wieder.

- **Double touch**

  Unter einem double touch versteht man zwei kurz hintereinander auftretende Berührungen des Displays. Die Bedeutung eines double touch kann von Applikation zu Applikation variieren. Eine gängige Reaktion wäre ein Zoom auf den berührten Bereich, sofern das sinnvoll ist.

- **Long press**

  Auf herkömmlichen Computern sehr selten verwendet, sind “long press Gesten” in mobilen Applikationen durchaus oft zu finden. Dabei verbleibt ein Benutzer für eine gewisse Zeit (meist rund eine Sekunde) mit dem Finger stationär auf einer Fläche. Wie auch beim double touch kann die Bedeutung variieren. Gängig wären aber in den meisten Fällen Kontext Menüs, die bei einem rechts-Klick auf einem Desktop System aufgerufen würden.

- **Pinch to zoom**

  Bei einer Berührung mit zwei Finger, die sich voneinander wegbewegen bzw. aufeinander zubewegen, erwarten Benutzer von mobilen touch-basierten Geräten meist, dass sich die Fläche unter den Fingern vergrößert oder verkleinert.

3.2 Grundlegende Visualisierungen für ein generisches Visualisierungs-Framework

Wenn man zu recherchieren beginnt wie viele verschiedene Möglichkeiten es gibt um Daten zu visualisieren, merkt man schnell, dass die Anzahl sehr unüberschaubar wird. In Abbildung 3.2 ist zumindest ein kleiner Teil davon in Form eines Periodensystems abgebildet. Die Grafik bietet einen schönen Überblick über die gängigsten Visualisierungen.

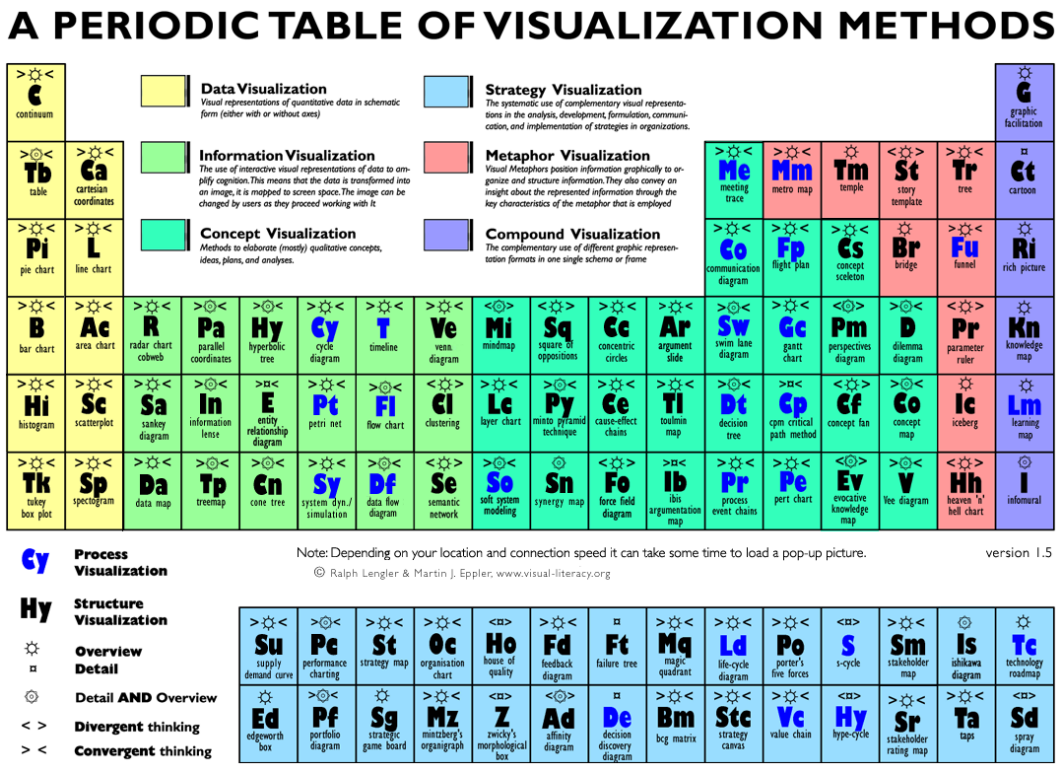

Abbildung 3.2: Das Periodensystem für Visualisierungen. Quelle: [Lengler und Eppler, 2012]

Der Grund warum die Zahl an verschiedenen Visualisierungen so hoch ist, ist vor allem darin begründet, dass es unzählige maßgeschneiderte Lösungen gibt, um einen ganz speziellen Datensatz möglichst übersichtlich abzubilden. Aber nicht jede Visualisierung eignet sich für jede Art von Daten. Aus diesem Grund ist es immer gut zu wissen, welche Daten man darstellen will. In Kapitel 2.1 wurde bereits beschrieben was die relevanten Daten sind, die im Zuge dieser Masterarbeit visualisiert werden.

Wenn man aber beispielsweise die Visualisierungen in Abbildung 3.2 genauer betrachtet, so kann man sehen, dass es viele gemeinsame Grundzüge bei den verschiedenen Vi-

sualisierungen gibt. Das ist nicht nur der in Abbildung 3.3 beschriebene Weg, wie man von Daten zu einer fertigen Darstellung kommt, sondern auch die graphische Aufbereitung.

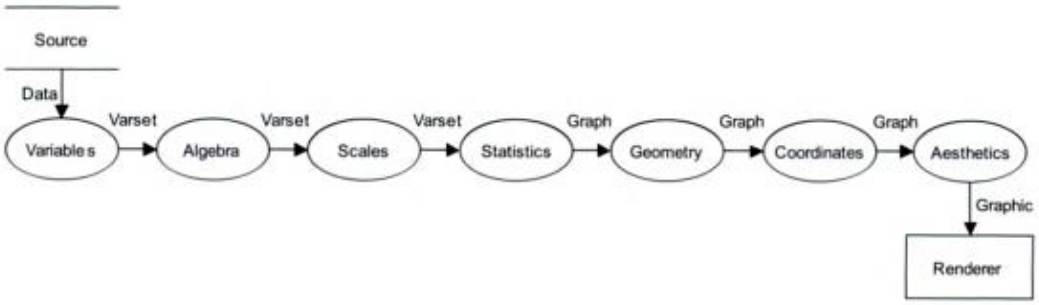

Abbildung 3.3: Von Daten zur Visualisierung. Quelle: [Wilkinson, 2005, S. 24]

Je nach Datensatz wird man sich für eine Visualisierung entscheiden, die die jeweiligen Daten am aussagekräftigsten darstellen kann. Im Folgenden werden gängige Visualisierungen beschrieben und erklärt, wofür sie sich gut eignen.

3.2.1 Darstellung von prozentuellen Anteilen

Hat man einen Datensatz, der eine prozentuelle Verteilung von Daten darstellt, so gibt es einige Visualisierungen die sich für eine gute Darstellung perfekt eignen. Die wohl bekannteste Möglichkeit ist aber das Tortendiagramm (oder auch Pie Chart) wie Abbildung 3.4 beispielhaft zeigt.

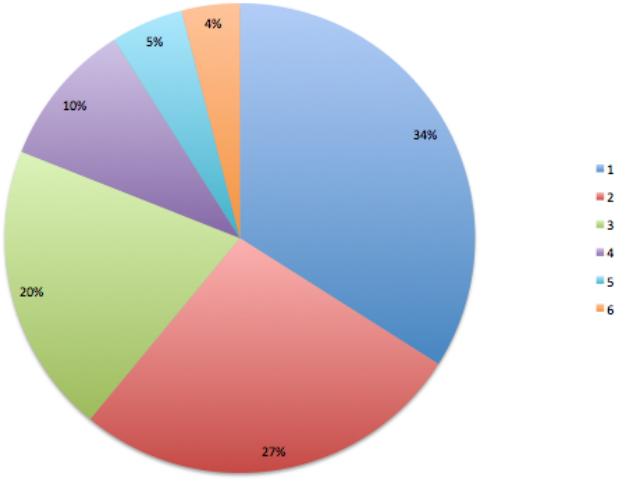

Abbildung 3.4: Beispiel für ein Tortendiagramm

Ein Tortendiagramm ist ein Kreis, der in so viele Segmente geteilt wird, wie es Einträ-

ge in dem dargestellten Datensatz gibt. Dabei ist die Größe eines jeden unterteilten Stücks abhängig davon, wie viele Prozent vom Gesamten der jeweilige Anteil ausmacht. Tortendiagramme eignen sich hervorragend zur Darstellung von relativ wenigen Werten. Sobald ein Tortendiagramm aber zu viele einzelne Segmente darstellen würde, ist es nicht mehr wirklich gut lesbar, da die einzelnen Segmente sehr klein würden. In diesem Fall sollte man sich überlegen, ob man die darzustellenden Werte nicht besser in einer anderen Visualisierung, wie beispielsweise einem Säulen oder Balken Diagramm, darstellt. Beide werden im nächsten Kapitel genauer beschrieben. Basierend auf dem Grundprinzip, wie ein Tortendiagramm funktioniert, gibt es nun dutzende verschiedene Varianten ein solches Diagramm darzustellen. Meist sind es aber nur graphisch unterschiedliche Formen der Darstellung.

3.2.2 Darstellung von absoluten Werten

Ist nicht unbedingt die prozentuelle Aufteilung von Werten entscheidend, sondern deren absolute Größe, so eignen sich Visualisierungen wie Säulen und Balken Diagramme sehr gut für eine übersichtliche Darstellung. Absolute Datensätze könnte man prinzipiell auch in einem Tortendiagramm darstellen, nur kann das schnell unübersichtlich werden, wenn man sehr viele Werte hat. In einem Tortendiagramm würde man dann sehr viele, sehr kleine Tortenstücke haben, die nicht mehr wirklich interpretierbar sind. Aus diesem Grund eignen sich für eine große Zahl an absoluten Werten Balken- und Liniendiagramme, wie in Abbildung 3.5 beispielhaft zu sehen, sehr gut.

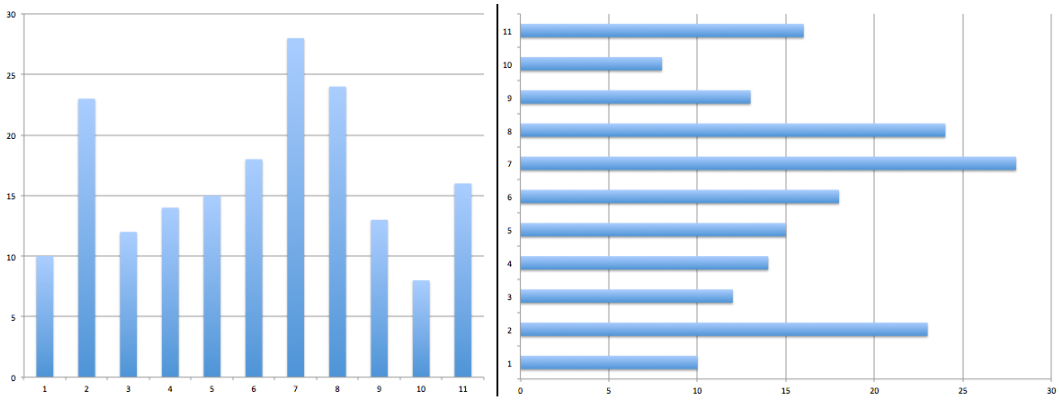

Abbildung 3.5: Links: Beispiel für ein Säulendiagramm
Rechts: Beispiel für ein Balkendiagramm

Im Prinzip ist ein Säulendiagramm und ein Balkendiagramm die gleiche Visualisierung, nur dass beim Säulendiagramm die einzelnen Werte entlang der X-Achse aufgetragen wer-

den und die Y-Achse für das Ablesen des entsprechenden Wertes verantwortlich ist und beim Balkendiagramm das Ganze genau umgekehrt ist. Je nachdem ob man bei der Darstellung mehr horizontalen oder vertikalen Platz zur Verfügung hat, entscheidet man sich für die eine oder andere Form der Darstellung.

3.2.3 Darstellung von Verläufen

Neben Datensätzen, die absolute oder prozentuelle Werte abbilden, gibt es auch welche, die Informationen über den Verlauf von Werten beinhalten. Um einen solchen Datensatz darzustellen, muss mindestens eine weitere Dimension, meist eine Zeitachse, eingeführt werden. Die wohl bekannteste und am Häufigsten verwendete Visualisierung für genau diesen Zweck, ist das Liniendiagramm. Ein Beispiel dafür ist in Abbildung 3.6 zu sehen.

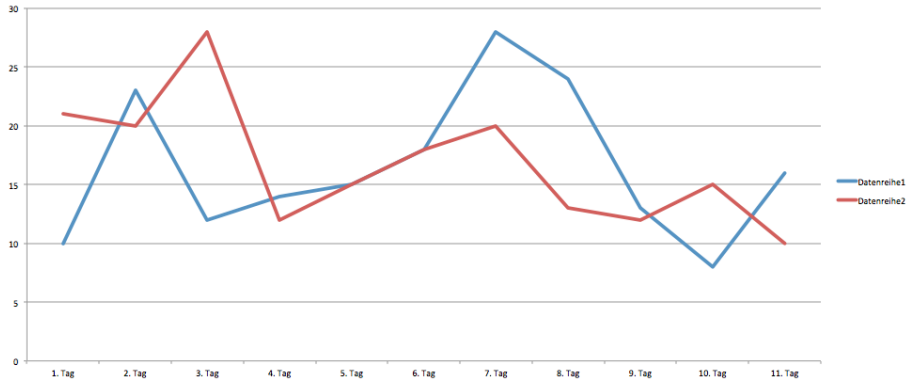

Abbildung 3.6: Beispiel eines Liniendiagramms

Bei der Darstellung eines Liniendiagramms wird jeweils ein Datensatz in einer Linie dargestellt. An der X-Achse sind die einzelnen Punkte abgebildet, an denen ein Wert vorliegt und an der Y-Achse werden dann entsprechend die jeweiligen Werte aufgetragen. Verbindet man nun die so entstandenen Punkte, so erhält man eine Linie.

Ein Liniendiagramm kann natürlich mehr als nur einen Datensatz, wie in Abbildung 3.6 zu sehen, darstellen. In diesem Fall wird pro darzustellendem Datensatz eine weitere Linie eingeführt. Hier sollte man allerdings darauf achten, dass es nicht zu viele Linien werden und vor allem, dass die Wertebereiche der einzelnen Linien in einem ähnlichen Bereich liegen. Wie in Kapitel 5.4 beschrieben, wäre es schwer die Änderungen eines Datensatzes zu erkennen, wenn die Werte des zweiten Datensatzes um ein Vielfaches abweichen. Eine Lösung wäre natürlich die Einführung einer weiteren Y-Achse, ob die Lesbarkeit des Diagramms dadurch allerdings besser wird, sei dahingestellt.

3.2.4 Darstellung geographischer Informationen

Wie bereits beschrieben, sind mögliche darzustellende Werte bei der Visualisierung von mobilen Sensordaten, unter anderen GPS Daten. GPS Daten zeichnen sich dadurch aus, dass sie mittel geographischer Breite und Länge einen Punkt auf der Erdoberfläche zweidimensional bzw. sogar dreidimensional, wenn man auch noch die geographische Höhe berücksichtigt, darstellen. Aus diesem Grund eignen sich keine der bereits vorgestellten Visualisierungen dafür, geographische Informationen abzubilden.

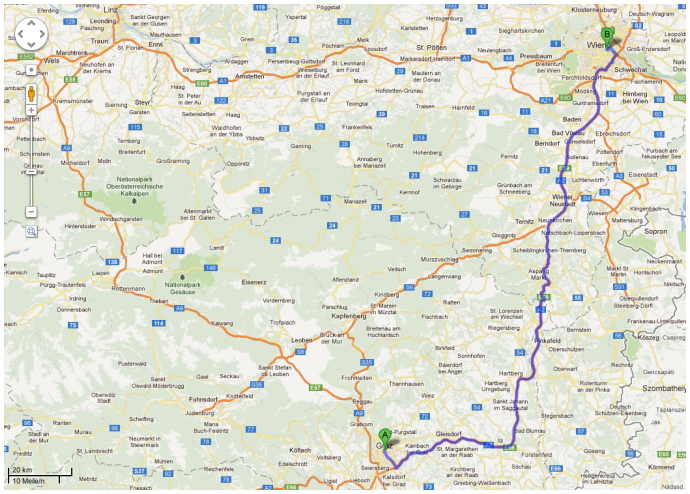

Abbildung 3.7: Beispiel für die Darstellung von geographischen Informationen Anhand einer Karte. Quelle: Google Maps

Die gängigste und dadurch wohl am besten zu verstehende Möglichkeit solche GPS Informationen darzustellen, ist eine Landkarte wie sie beispielsweise in Abbildung 3.7 zu sehen ist. Punkte, die durch geographische Längen- und Breiteninformation beschrieben sind, können in ein Koordinatensystem umgerechnet werden, dass dem angezeigtem Kartenausschnitt entspricht und somit über der darunterliegenden Karte eingeblendet werden. Abbildung 3.7 zeigt beispielsweise eine Informationen für eine mögliche Autostrecke von Graz nach Wien.

Zeitliche Information ist auf einer Karte nicht gut darstellbar. In Kapitel 5.4 bzw. 5.5 wird eine Möglichkeit vorgestellt, die es dem Benutzer erlaubt, mittels Interaktion mit anderen Visualisierungen, die über eine zeitliche Komponente verfügen, diese Information auch in einer Karte darzustellen.

Kapitel 4

Ähnliche Arbeiten

"Einmal sehen ist besser als zehnmal hören." *deutsches Sprichwort*

Ein Charakteristikum, das sich die meisten existierenden anwenderorientierten Applikationen teilen, ist die Tatsache, dass sie Daten auf die eine oder andere Wiese darstellen. Seien diese Daten Wirtschaftszahlen, die in Liniendiagrammen den Verlauf von Aktien darstellen, oder Bilder, die Erinnerungen an den letzten Urlaub auffrischen lassen. Jedes Spiel, dass man auf einem Computer spielt, ist im Prinzip nichts anderes, als das schön verpackte Darstellen von Zahlen. Ähnliches machen auch Produktivitätsanwendungen wie Tabellenkalkulationen, in denen versucht wird, Zahlen in leichter verständlichen Visualisierungen, wie beispielsweise Linien- und Balkendiagramme, darzustellen. Für den Menschen ist es in der Regel einfacher solche Darstellungen zu verarbeiten, als sich durch lange Listen mit Zahlen zu kämpfen. Somit sollte jedes Computerprogramm versuchen dem Benutzer durch möglichst passende Visualisierungen entgegenzukommen.

Um mobile Sensordaten dem Benutzer möglichst einfach und verständlich darzustellen, beleuchtet dieses Kapitel zuerst einige ähnliche Arbeiten, die es in folgenden Bereichen gibt:

- Mobile Sensordaten
- Gesundheitsrelevante Daten
- Self-Tracking (elektronische Tagebücher)
- User-Context Daten
- Effizientes Darstellen von großen Datenmengen

Am Ende des Kapitels werden die einzelnen vorgestellten Programme miteinander verglichen.

4.1 Visualisierung von mobilen Sensordaten

Da moderne mobile Geräte über immer mehr Sensoren verfügen und diese Masterarbeit die Visualisierung von genau diesen Daten zum Ziel hat, stellt dieses Unterkapitel drei Applikationen vor, die mobile Sensordaten darstellen.

4.1.1 Sensor Monitor

Sensor Monitor[1] ist eine Open Source iOS Applikation, die alle in einem iPhone bzw. iPad eingebauten Sensoren ausliest und visualisiert. GPS Daten werden einerseits mittels Zahlen dargestellt, aber auch auf einer Karte eingezeichnet. Werte, die der Bescheinigungs- bzw. Gyroskopsensor liefert, werden einerseits als Zahlen, aber auch in einem Live-Liniendiagramm visualisiert. Die Darstellung des Kompass setzt gleich wie die des Touchsensors und des Lautstärkemessers rein auf Zahlen.

Weiters ist zu erwähnen, dass Sensor Monitor keine der aufgezeichneten Daten speichert. Die Applikation sieht sich somit rein als Visualisierung von live Daten.

Abbildung 4.1: Screenshot der Applikation Sensor Monitor

[1] https://sites.google.com/a/fuzzface.net/app//sensor-monitor (zuletzt besucht am 15. März 2012)

4.1.2 SensorLog

Ähnlich wie Sensor Monitor, liest SensorLog[2] alle verfügbaren Sensor Daten eines iOS Gerätes aus und stellt diese dar. Bei der Visualisierung wird vor allem auf Live-Liniendiagramme gesetzt, aber auch auf eine Karte für GPS Informationen.

Im Gegensatz zur Sensor Monitor Applikation, bietet SensorLog zumindest eine Möglichkeit Daten zu exportieren, wenngleich auch nicht diese zu speichern.

Abbildung 4.2: Screenshot der Applikation SensorLog

4.1.3 xSensor

Mit weniger verschiedenen Visualisierungen, aber auch weniger ausgelesener Sensoren kommt xSensor[3] aus. Die Anwendung, die es für iPhone und iPad gibt, beschränkt sich auf die textuelle Darstellung von GPS Informationen. Bei der Anzeige der Werte vom digitalen Kompass und Beschleunigungssensor setzt xSensor, wie auch SensorLog bzw. Sensor Monitor auf ein Live-Liniendiagramm.

[2] http://www.berndthomas.net/berndthomas.net/SensorLog.html (zuletzt besucht am 15. März 2012)

[3] http://www.xbow.com/asset-tracking/support/index.html (zuletzt besucht am 15. März 2012)

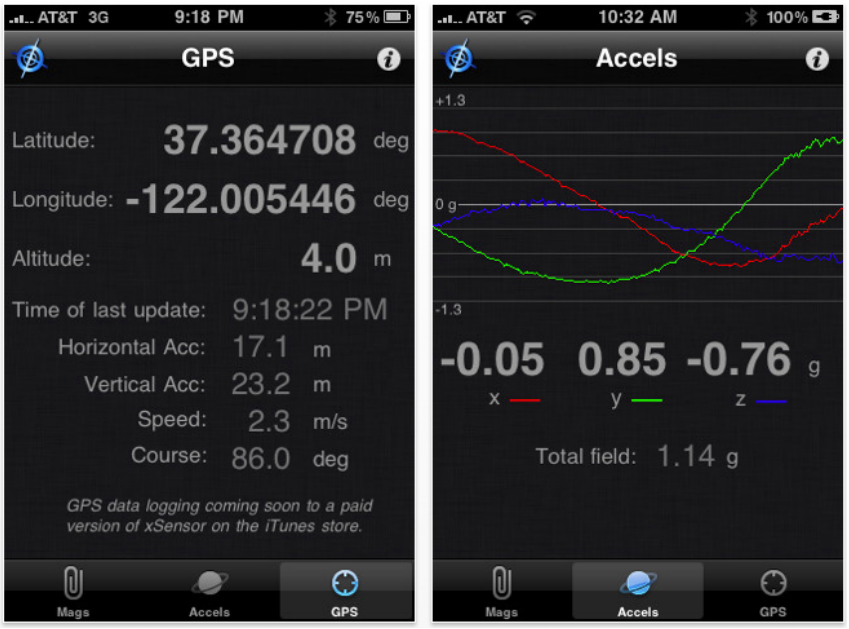

Abbildung 4.3: Screenshot der Applikation xSensor

4.2 Visualisierung von gesundheitsrelevanten Daten

Gerade im Gesundheitsbereich kann ein Mobiltelefon dabei helfen, gesundheitsrelevante Daten für eine bessere Analyse des behandelnden Arztes, aufzuzeichnen. Auch für diesen Zweck gibt es diverse Programme, die dabei unterstützend wirken können. In diesem Unterkapitel werden zwei Arbeiten vorgestellt, die sich mit diesem Thema beschäftigen.

4.2.1 LifeLog

LifeLog[4] ist eine medizinische Anwendung um gesundheitsrelevante Daten wie Schmerz, Stress, Strapazen, Glück, etc. zu erfassen aber auch zu visualisieren. Die Erfassung passiert in der LifeLog Applikation aber nicht über Sensoren, sondern rein über Benutzer Inputs mittels einer 0-10 Bewertung von diversen Fragen. Für die Auswertung dieser Zahlen setzt die Applikation auf bekannte Darstellungsvarianten wie Linien- und Balkendiagramme. Im Großen und Ganzen ist die Anwendung aber sehr zahlenlastig.

4.2.2 BeWell

Eine weitere Arbeit im Bereich der Visualisierung von gesundheitsrelevanten Daten ist “BeWell: A Smartphone Application to Monitor, Model and Promote Wellbeing” [Lane et al.,

[4] http://www.lifelogapp.com/ (zuletzt besucht am 15. März 2012)

Abbildung 4.4: Screenshots der LifeLog Applikation

2011]. Im Gegensatz zu anderen Arbeiten auf diesem Sektor, die stark auf Daten von externen Sensoren setzten, ging es den Forschern hinter BeWell darum, mit den Smartphone-internen Sensoren auszukommen. Sie haben eine auf dem Android Betriebssystem basierte Anwendung erstellt, die Daten wie Schlaf, körperliche Ertüchtigung sowie soziale Interaktionen aufzeichnet und auch visualisiert. Als Feedback für den Anwender, setzt BeWell bei der Darstellung dieser Daten, wie in Abbildung 4.5 zu sehen, auf eine maßgeschneiderte Visualisierung. Dabei handelt es sich um Meeresbewohner, die je nach Menge in der sie am Bildschirm auftreten und deren Aktivität, die gesundheitlichen Eigenschaften des Anwenders widerspiegeln sollen. Weiters werden die aufgezeichneten Werte auch durch Text dargestellt.

Abbildung 4.5: Screenshots aus BeWell: A Smartphone Application to Monitor, Model and Promote Wellbeing [Lane et al., 2011]

4.3 Visualisierung von Self-Tracking Applikationen (elektronische Tagebücher)

Neben der Möglichkeit reinen Text niederzuschreiben, bieten elektronische Tagebücher unter anderem die Möglichkeit Bilder und geographische Informationen auf einfachen Weg in den Text zu integrieren. Wie diverse Programme diese Daten visualisieren, wird in diesem Unterkapitel anhand von zwei Apps gezeigt.

4.3.1 Trip Journal

Trip Journal[5] rühmt sich selbst die beste Reise-Tagebuch App zu sein und ist sie auf allen derzeit gängigen mobilen Plattformen wie iOS, Android, Bada und Symbian vertreten. Die Applikation dient als Tagebuch speziell für Reisen, indem sie mittels dem im Mobilgerät verbauten GPS-Empfänger Wege speichert, dazu Bilder georeferenziert speichert und natürlich auch das Schreiben von Notizen erlaubt. Bei den Visualisierungen setzt Trip Journal dabei hauptsächlich auf Karten sowie Bilder.

[5] http://www.trip-journal.com (zuletzt besucht am 15. März 2012)

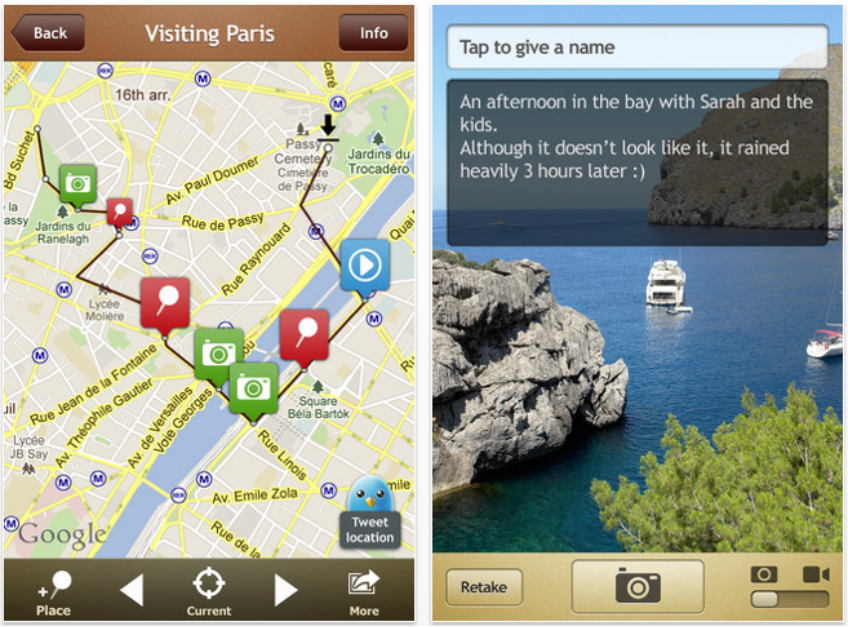

Abbildung 4.6: Kartenvisualisierung mit eingeblendeten Informationen zu Sehenswürdigkeiten von Trip Journal

4.3.2 Maxjournal

Ähnlich wie das eben angesprochene Trip Journal ist Maxjournal[6] eine Tagebuch Applikation für iOS Geräte. Es setzt allerdings keinen Schwerpunkt auf einen speziellen Einsatzbereich, sondern ist sehr generell gehalten. Somit sind Karten im Gegensatz zu Trip Journal nicht die am Häufigsten anzutreffende Visualisierung, sondern eher Bilder und niedergeschriebene Texte.

Interessant ist vor allem die Kalenderfunktion von Maxjournal. Im Gegensatz zu kompakten Datepicker Komponenten, ist der Kalender in Maxjournal über den gesamten Bildschirm verteilt und ermöglicht so eine recht einfache Navigation.

[6] http://www.omaxmedia.com/ (zuletzt besucht am 15. März 2012)

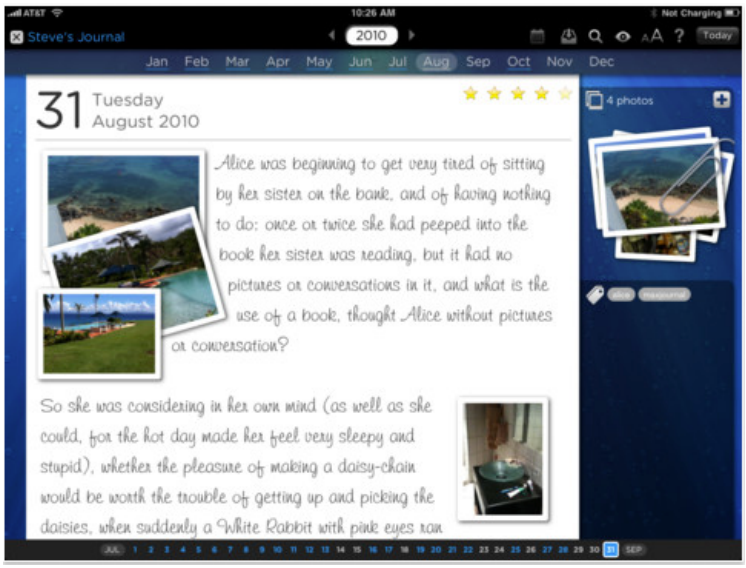

Abbildung 4.7: Ein Eintrag in Maxjournal mit Bildern

4.4 Visualisierung von User-Context Daten

Da die für diese Masterarbeit aufgezeichneten mobilen Sensordaten großteils Aufschluss über den User-Context [Abowd et al., 1999] geben sollen, zeigt dieses Unterkapitel anhand von vier Arbeiten in diesem Bereich, wie die Darstellung der ermittelten User-Context Daten realisiert werden kann.

4.4.1 Affective Diary

Das Affective Diary [Ståhl und Höök, 2006, S. 373] ist ein Versuch eines virtuellen Tagebuchs um reflektives Lernen [Boud, 1985] zu unterstützen. Dabei ging es den Autoren vor allem um die Darstellung von Gefühlen. Das passiert aber weder mit konventionellen Visualisierungen, wie Linien oder Balkendiagrammen, noch mit etwas spezielleren Darstellungsmethoden wie einer Moodmap (siehe Kapitel 5.3.4), sondern, wie in Abbildung 4.8 zu sehen, in Form einer Figur die je nach Lage und Farbe ein anderes Gefühl verkörpern soll. In Affective Diary wird aber auch auf z.b. Text zurückgegriffen, um Emails und SMS anzuzeigen, die für einen Gefühlszustand verantwortlich sein können.

4.4.2 MobiVis

Eine weitere Arbeit, die sich mit dem Visualisieren von User-Context Daten [Abowd et al., 1999] [Rath, 2010] beschäftigt, ist MobiVis [Zeqian und Kwan-Liu, 2008]. Bei Mo-

Abbildung 4.8: Visualisierungen aus Affective Diary [Ståhl und Höök, 2006, S. 373]

biVis handelt es sich um ein Forschungsprojekt, mit dem Ziel ein visuelles Analyse Tool bereitzustellen. Es werden vor allem soziale Zusammenhänge von mehreren Personen visualisiert. Um das möglichst intuitiv zu erreichen, wird dafür das Konzept von sogenannten "Behavior Rings" (siehe Abbildung 4.9) eingeführt um die Interaktion von Personen bzw. Personengruppen miteinander darzustellen.

Die Daten, die in dem MobiVis Paper dargestellt werden, stammen aus dem MIT Reality Mining dataset[7] und beziehen sich nicht direkt auf die Daten einer einzelnen Person, sondern auf mehrere Personen.

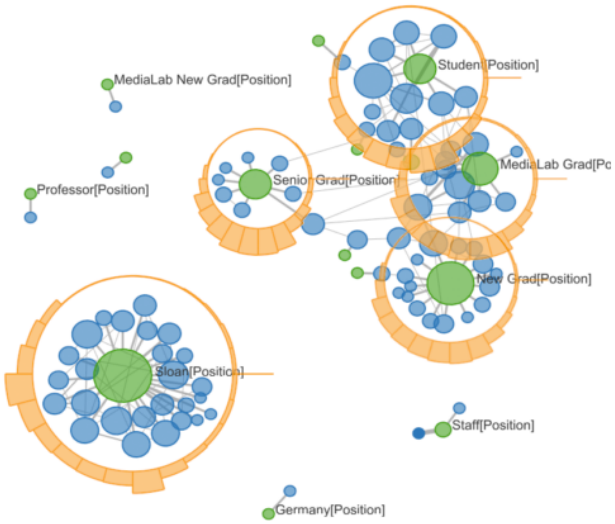

Abbildung 4.9: "Behaviour Rings" aus [Zeqian und Kwan-Liu, 2008, S. 181]

[7] http://reality.media.mit.edu/dataset.php (zuletzt besucht am 16. März 2012)

4.4.3 Moodmap App

Im Zuge des EU-Projektes Mirror[8] ist unter anderem die Moodmap App[9] entstanden. Dabei handelt es sich um eine Web-Applikation, die das Erfassen des eigenen Gefühlszustandes mit Hilfe einer Moodmap erlaubt. Eine genauere Beschreibung der Funktionsweise einer Moodmap folgt in Kapitel 5.3.4, da diese auch im VisualizationKit-Framework, dass im Zuge dieser Masterarbeit entstanden ist, implementiert wurde.

Die hier vorgestellte Moodmap Applikation kann aber neben dem Aufzeichnen von Gefühlszuständen, diese weiters noch mit Notizen versehen und vor allem mit anderen Personen vergleichen. So ist es beispielsweise möglich, für eine geschäftliches Meeting eine gemeinsame Moodmap Session zu starten, um anschließend gemeinsam darüber reflektieren zu können [Russell, 1980] [Itten, 2003] bzw. [Ståhl et al., 2005] [Boud, 1985].

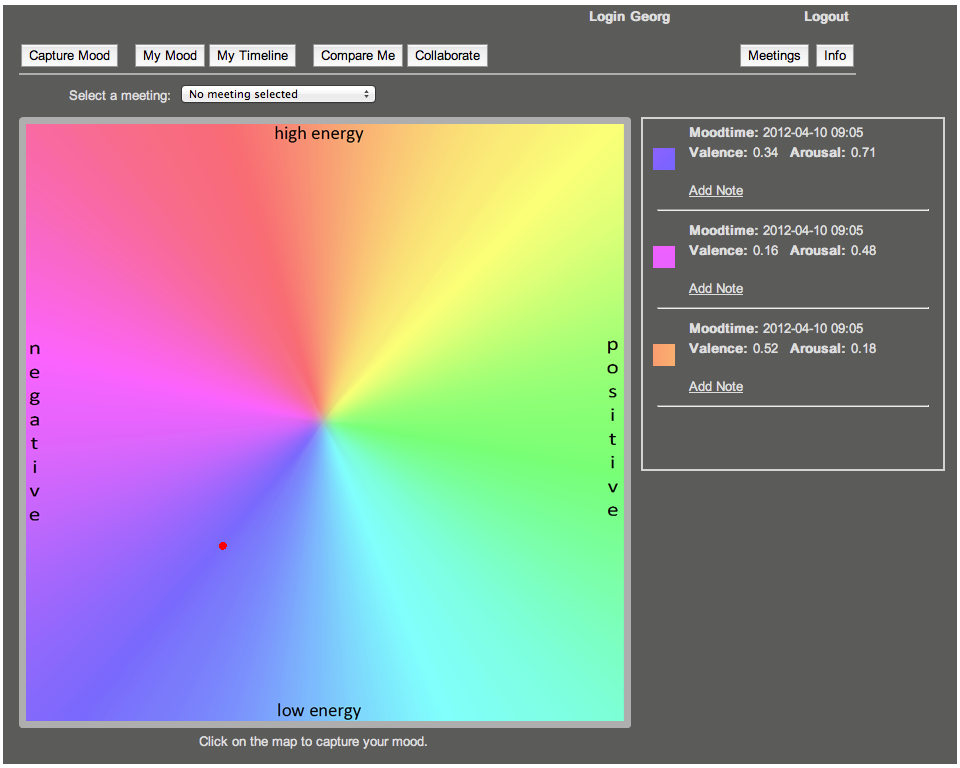

Abbildung 4.10: Screenshot der MoodMap App

[8] http://www.mirror-project.eu/ (zuletzt besucht am 2. April 2012)

[9] http:moodmap.know-center.tugraz.at/ (zuletzt besucht am 2. April 2012)

4.4.4 TD App

Eine weitere Applikation, die im Zuge des EU-Projektes Mirror[10] entstanden ist, ist die TD-App (Abbildung 4.11). Auch hier handelt es sich um eine Webapplikation, die es ermöglicht, aufgezeichnete User-Context Daten zu visualisieren. Als Back-End für die TD App dient das KnowSe Framework [Rath, 2010], das auch am Know-Center [11] entstanden ist. Desktop Sensoren zeichnen die Aktivitäten eines Benutzers beim Arbeiten am Computer auf und speichern sie im KnowSe. Diese Daten werden von der TD App abgefragt, um sie chronologisch in einer kompakten Übersicht darzustellen. Weiters kann der Benutzer für jeden Eintrag weitere Details, wie das aktive Programm und welche Ressourcen dieses gerade verwendet hat, sehen. Bei den Visualisierungen setzt die TD-App dabei auf einen interaktiven Zeitstrahl, sowie auf diverse Diagramme, um Details darzustellen.

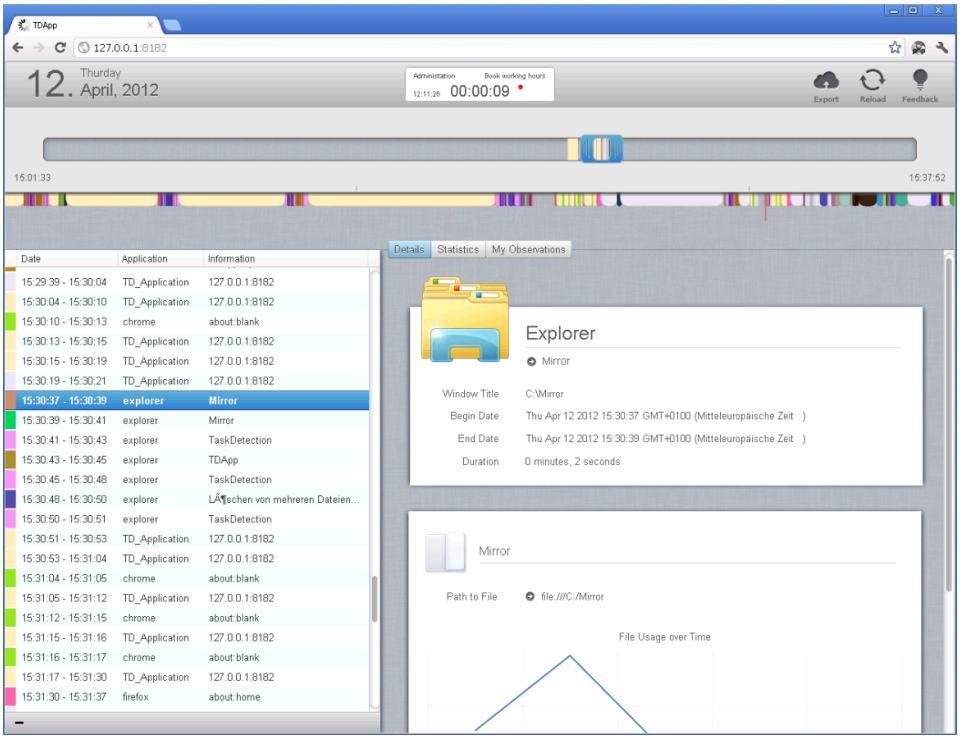

Abbildung 4.11: Screenshot der TD App

[10] http://www.mirror-project.eu/ (zuletzt besucht am 2. April 2012)

[11] http://know-center.tugraz.at/ueber-uns (zuletzt besucht am 12. August 2011)

4.5 Effizientes Visualisieren von großen Datenmengen

Die effiziente Darstellung von großen Datenmengen ist aus zwei Gründen nicht ganz trivial. Zum Einen ist die Anzahl an verfügbaren Pixeln auf einem Monitor begrenzt, zum Anderen kann es vor allem im mobilen Umfeld aufgrund von limitierter Hardwareleistung zu Performanceproblemen kommen, wenn man sehr viele Datensätze anzeigen will. Die Lösung von einigen Arbeiten werden in diesen Unterkapitel vorgestellt.

4.5.1 Extreme Visualization

"Extreme Visualization - Squeezing a Billion Records into a Million Pixels" von [Shneiderman, 2008] ist ein Paper, das sich mit der Visualisierung von sehr großen Datenmengen auf möglichst kleinem Raum beschäftigt. Dabei werden diverse Ansatzpunkte vorgestellt und in Kategorien unterteilt:

Visualisierung von nicht teilbaren Datensätzen:
Abbildung 4.12 zeigt ein Beispiel für Datensätze, die nicht mehr weiter teilbar sind. In diesem Fall handelt es sich um die Aufteilung von Verzeichnissen auf einem Fileserver. Da es sich um sehr viel Information handelt, die auf relativ wenig Raum dargestellt werden muss, bedient sich diese Visualisierung an Farben, verschiedenen Größen und Anordnungen der Flächen, um so diverse Informationen zu kodieren.

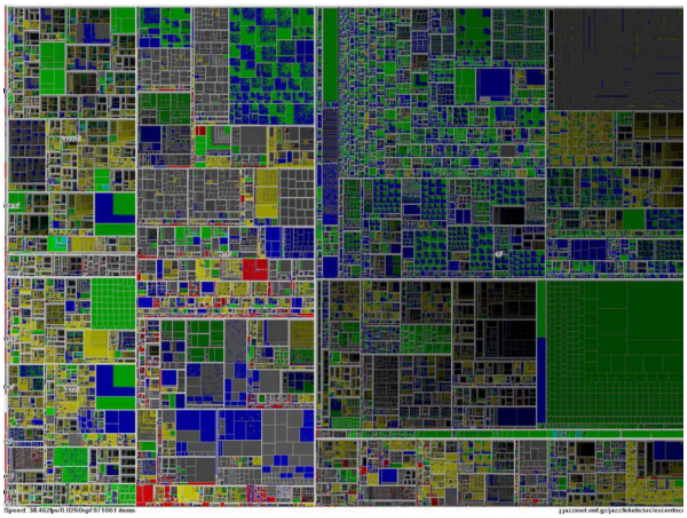

Abbildung 4.12: Verteilung von Verzeichnissen auf einem Fileserver. Quelle: [Shneiderman, 2008]

Visualisierung von aggregierten Datensätzen:

Um noch mehr Information auf der gleichen Fläche darstellen zu können, muss eine Vorverarbeitung von Daten stattfinden. Ein Beispiel das in [Shneiderman, 2008] gezeigt wird, ist in Abbildung 4.13 zu sehen. Hier handelt es sich um eine Auswertung von Baseball Spieler Statistiken mittels des Programms Spotfire[12].

Nach einigen hundert Spielen haben die Spieler sehr viele Schläge getätigt und dabei hoffentlich auch den ein oder anderen Homerun erzielen können. Um diese Daten mehrerer Spieler noch einigermaßen übersichtlich miteinander vergleichen zu können, bedarf es einer gewissen Aggregation der Daten, bevor diese dargestellt werden können. Anders wäre es wohl nicht möglich auf einem gewöhnlichen Bildschirm die gesamte Information unterzubringen. Nach dieser Vorberechnung der Datensätze ist es mit einfachen Visualisierungen, wie Liniendiagrammen und Punktwolken, möglich, dem Benutzer übersichtliche Statistiken anzubieten.

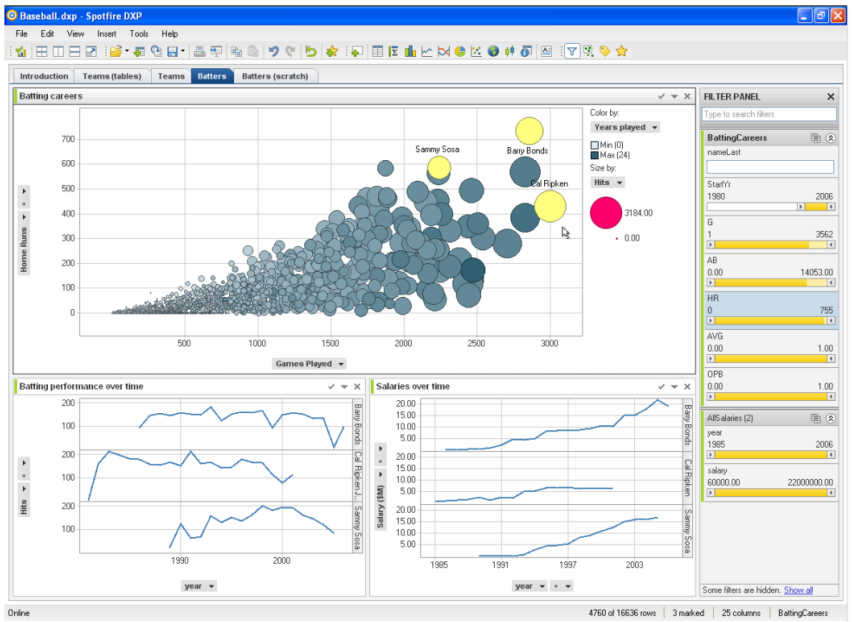

Abbildung 4.13: Visualisierung von Baseball Statistiken durch das Programm Spotfire. Vorgestellt in [Shneiderman, 2008]

Visualisierung von Häufungen:

Wenn die Menge an Daten noch größer wird, interessiert es den Benutzer unter Umständen gar nicht mehr einzelne Werte exakt abzulesen zu können. Viel entscheidender ist es dann einen Gesamtüberblick über die Datensätze zu erhalten. In Abbildung 4.14 kann man das an

[12] http://spotfire.tibco.com/ (zuletzt besucht am 5. April 2012)

dem Beispiel von fatalen Unfällen sehen, die in einer Parallelen Koordinaten Visualisierung dargestellt werden. Durch effizientes Clustering (im Abbildung 4.14 rechts zu sehen) ist es so für den Benutzer möglich, relevante Informationen zu extrahieren.

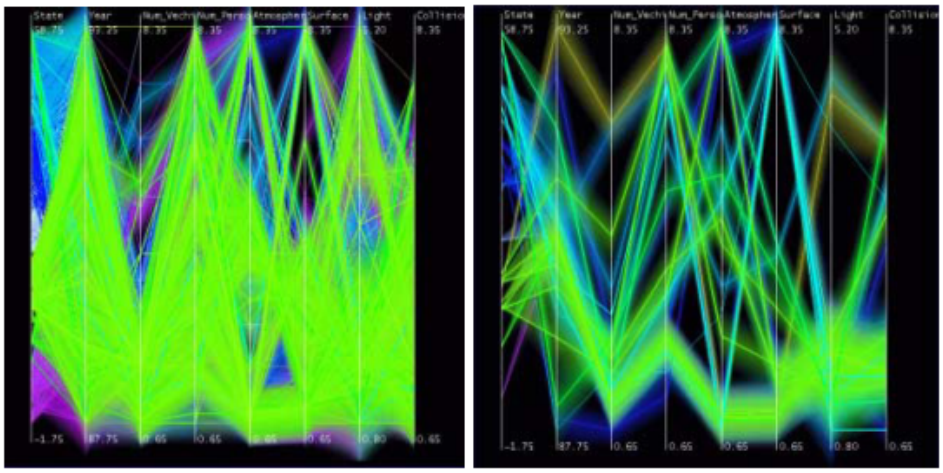

Abbildung 4.14: Die Darstellung von 230.000 Datensätzen über fatale Unfällen in einer Parallelen Koordinaten Visualisierung. Quelle: [Shneiderman, 2008]

4.5.2 Literature Fingerprinting

Eine weitere sehr spannende Arbeit, die sich mit der Visualisierung von sehr großen Datenmengen beschäftigt, ist "Literature Fingerprinting: A New Method for Visual Literary Analysis" [Keim und Oelke, 2007]. In dieser Arbeit wird ein Framework namens LiteratureVis vorgestellt, das bei der Analyse von Textdokumenten helfen soll.

Meistens wird Text, wenn dieser analysiert wird, nur anhand eines einzigen Features kategorisiert. Um aber mehr Information über das Dokument zu erhalten, macht es Sinn, sich mehrere Features anzuschauen. LiteratureVis ist eine interaktive Visualisierung die das ermöglicht. In Abbildung 4.15 ist beispielsweise eine Visualisierung der Bibel zu sehen. Dabei entspricht jeder Pixel einem Vers und durch die Farbe wird die Länge dieses Verses dargestellt. So kann man schön auf einen Blick sehen, dass die einzelnen Kapitel der Bibel sehr unterschiedliche Eigenschaften haben. Das deckt sich auch schön mit dem Fakt, dass es tatsächlich mehrere Autoren gab, die an der Bibel geschrieben haben.

Das LiteratureVis Framework kann aber auch gut eingesetzt werden, um die Lesbarkeit

Abbildung 4.15: Visuelle Darstellung der Bibel. Quelle: [Keim und Oelke, 2007]

von Texten zu analysieren und zu verbessern. Sind sehr viele Stilwechsel in einem Werk vorhanden, ist es für den Leser oft schwierig sich immer wieder neu einzustellen. Solche Schwächen kann eine entsprechende Visualisierung recht einfach aufdecken.

4.5.3 Roambi Visualizer

Der Roambi Visualizer[13] ist eine iPhone/iPad Applikation, die Wirtschaftsdaten auf eine sehr ansprechende Art und Weise visualisiert. Es gibt für die meisten Wirtschaftsdaten eine Vielzahl an möglichen Darstellungsformen. Die meisten erlauben auch eine direkte Interaktion, um Werte noch besser ablesen zu können und den Benutzer zum Spielen und somit zum besseren Kennenlernen der Applikation zu animieren.

Obwohl der Roambi Visualizer keine mobile Sensordaten darstellt, wird diese Applikation hier vorgestellt, da diese es sehr gut schafft das Interesse des Benutzers mit seinen Visualisierungen auf sich zu ziehen. Das in dieser Masterarbeit entstandene Torten-Diagramm,

[13] http://www.roambi.com/ (zuletzt besucht am 10. Juni 2011)

mit der Möglichkeit sich zu drehen, ist von dieser App inspiriert worden.

Abbildung 4.16: Linien- und Tortendiagramm Darstellung von Roambi Visualizer

4.5.4 Core Plot

Bei dem CorePlot Projekt[14] handelt es sich um ein Open-Source Framework zur Visualisierung von Daten mittels Graphen und Charts. Der Grund warum im Zuge dieser Masterarbeit ein eigenes Visualisierungs-Framework erstellt wurde ist der, dass das CorePlot Framework zu langsam war, um Daten auch in Echtzeit anzuzeigen. Es diente aber als Inspiration wie das in dieser Masterarbeit entwickelte Framework implementiert wurde.

[14] http://code.google.com/p/core-plot/ (zuletzt besucht am 10. Juni 2011)

4.6 Vergleich

Wie die einzelnen Arbeiten, die in diesem Kapitel vorgestellt wurden, gezeigt haben, gibt es unzählige Varianten um Daten effizient darzustellen. Abbildung 4.17 soll dabei helfen, die Charakteristika der einzelnen Arbeiten besser zu veranschaulichen.

	Visualisierungen				
	Text	Bilder	Diagramme	Karte	selbst definiert
Visualisierung von mobilen Sensordaten					
Sensor Monitor	x		x	x	
SensorLog	x		x		
xSensor	x		x		
Visualisierung von gesundheitsrelevanten Daten					
LifeLog	x	x		x	
BeWell	x	x			x
Visualisierung von Self-Tracking Applikationen (elektronische Tagebücher)					
Trip Journal	x	x		x	
Maxjournal	x	x		x	
Visualisierung von User-Context Daten					
Affective Diary	x				x
MobiVis					x
Moodmap					x
TD App	x		x		x
Effizientes Visualisieren von große Datenmengen					
Extreme Visualization			x		x
Literature Fingerprinting					x
Roambi Visualizer	x		x		
Core Plot			x		

Abbildung 4.17: Vergleich der vorgestellten ähnlichen Arbeiten

Vor allem bei der genaueren Betrachtung der Arbeiten, die sich mit der Visualisierung von User-Context Daten beschäftigen, kann man sehen, dass diese Applikationen vor allem auf spezielle und genau maßgeschneiderte Visualisierungs-Lösungen setzen.

Andere Applikationen wiederum, die vor allem auf das Darstellen von mobilen Sensordaten, aber auch gesundheitsrelevanten Daten und Self-Tracking spezialisiert sind, setzen

dagegen hauptsächlich auf etwas generischere Darstellungsformen wie diverse Diagramme, Bilder, Text und Landkarten.

Wenn es darum geht größere Datenmengen darzustellen, ist keine ganz so starke Tendenz zu generischen Visualisierungsformen bzw. maßgeschneiderten Lösungen zu erkennen. Vor allem aber setzen diese Visualisierungen unter anderem auf die Unterstützung durch Farben und Formen.

Aus diesem Grund ist die Entscheidung für die Implementierung des Visualisierungs-Frameworks dieser Masterarbeit darauf gefallen, generische Visualisierungsformen mit den Vorteilen von maßgeschneiderten Lösungen zu vereinen.

Kapitel 5

Implementation

"All truths are easy to understand once they are discovered; the point is to discover them." *Galileo Galilei*

5.1 Einleitung

Der praktische Teil dieser Masterarbeit befasste sich mit der Erstellung eines Visualisierungs-Frameworks zur interaktiven und performanten Darstellung von mobilen Sensordaten auf mobilen Geräten. Als Plattform wurde iOS, das Betriebssystem von Apples mobilen Geräten iPhone, iPod Touch und iPad, gewählt.

Dieses Kapitel beschreibt die Lösung die im Zuge dieser Masterarbeit erarbeitet wurde im Generellen und geht dann auf die einzelnen Visualisierungen im Detail ein.

5.2 Übersicht

Der Hauptbestandteil der praktischen Arbeit an dieser Masterarbeit, war die Implementierung eines Visualisierungs-Frameworks zur interaktiven und performanten Darstellung mobiler Sensordaten auf dem mobilen Geräten selbst zu implementieren. Wie bereits in Kapitel 3 beschrieben, war der Ansatz dieser Arbeit, generische Darstellungsvarianten als Basis für mehrere verschiedene Visualisierungen zu verwenden, anstatt die gesamte Information in eine Darstellungsform zu packen. Da es, wie in Kapitel 2.1 beschrieben, sehr

viele verschiedene Daten gibt die dargestellt werden sollen, war es nötig viele verschiedene Visualisierungen dafür zur Verfügung zu stellen.

In den folgenden Kapiteln wird der Aufbau jeder einzelnen Visualisierungen beschrieben. Zuerst wird aber auf die generelle Architektur eingegangen, die alle Visualisierungen gemeinsam haben.

Grundsätzlich sind alle Visualisierungen von der Superklasse VKVisualization abgeleitet. Das Klassendiagramm dazu wird in Abbildung 5.1 gezeigt. Das VK im Klassennamen steht für VisualizationKit und dient als Prefix für alle in dieser Masterarbeit erstellten Klassen, die mit der Visualisierung zu tun haben. Das Prefixen von Klassen ist in Objective-C eine gängige Benamungskonvention, da es keine Namespaces wie in C++ oder Packages wie in Java gibt [Neuburg, 2011].

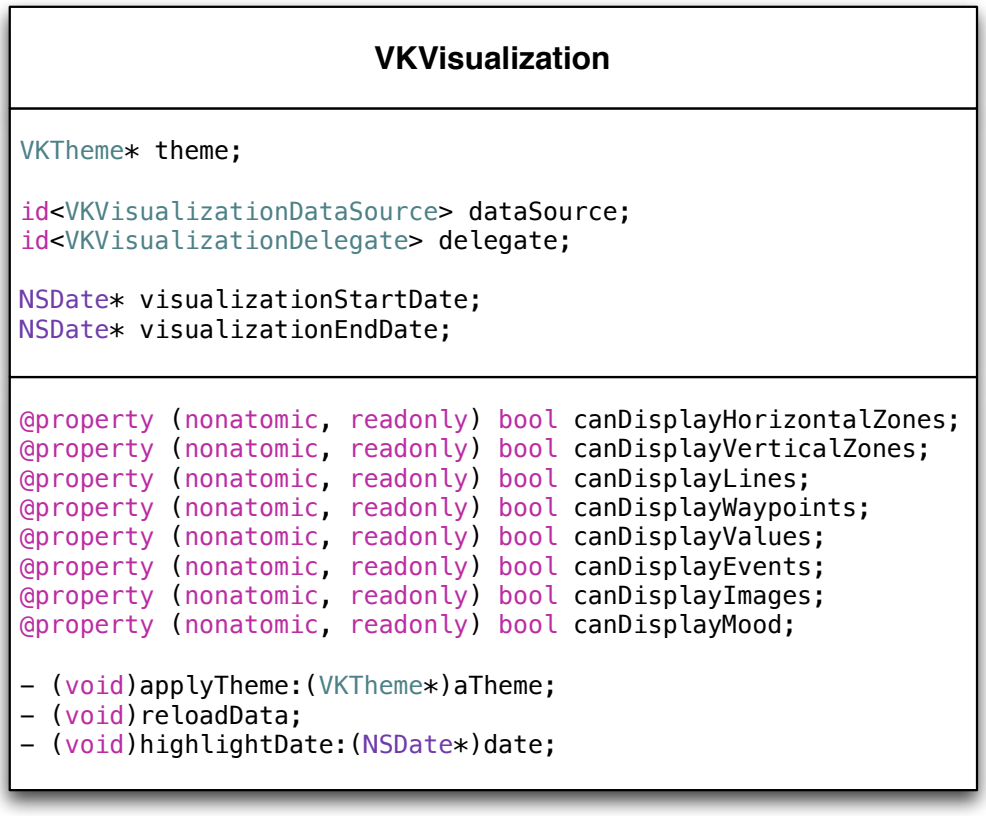

Abbildung 5.1: Klassendiagramm der Superklasse VKVisualization

Die Klasse VKVisualization selbst ist von UIControl [Apple, 2012f] abgeleitet, was durch wiederum seine Ableitung von UIView [Apple, 2012i] dazu führt, dass man darin eigenen Drawing Code schreiben kann. Jeglicher Code dazu wird in der Methode

```
1 - (void)drawInContext:(CGContextRef)ctx;
```

ausgeführt. Ein selbstständiges Auslösen von Drawing Code an einer anderen Stelle wäre sehr unperformant, da das Betriebssystem entscheidet, wann etwas gezeichnet werden muss. Wenn der Programmierer selbst dafür sorge tragen will dass sich eine UIView neu zeichnet, setzt man das Flag

```
1 - (void)setNeedsDisplay:(BOOL)flag;
```

auf TRUE und das Betriebssystem wird dann entscheiden, wann es tatsächlich gezeichnet werden soll. Das soll verhindern, dass sich z.b. während eines Schleifendurchlaufs, der etwas an der Darstellung eines Controls ändern würde, nicht jedesmal der komplette Zeichen Code durchlaufen werden muss, sondern erst dann, wann der UI-Thread wieder an der Reihe ist.

Da alle Visualisierungen über eine Theme-Klasse namens VKTheme gestylt werden, und darin beispielsweise der Hintergrund einer jeden Visualisierung beschrieben ist, ist die Superklasse VKVisualization für das Zeichnen all dieser Dinge zuständig. Somit macht jedes von VKVisualization abgeleitete Control gleich zu Beginn seines eigenen Drawing Codes einen Aufruf an die Supermethode.

```
1 - (void)drawInContext:(CGContextRef)ctx
2 {
3   [super drawInContext:ctx];
4   // Custom drawing code
5 }
```

Weiters soll eine jede Visualisierung seine Daten von einer definierten DataSource [Apple, 2012b] erhalten. Das DataSource Prinzip ist in der Programmierung mit Objective-C ein sehr gängiges Design Pattern. In einem Protokoll wird definiert welche Methoden eine Datenquelle implementieren muss, damit zum Beispiel eine Visualisierung an Daten kommt. Der Vorteil dieses Designpatterns gegenüber anderen Methoden Daten in die Visualisierung zu bekommen ist der, dass die Visualisierung wirklich erst zu dem Zeitpunkt, zu dem sie es selbst für wichtig hält Daten zu bekommen, danach fragen kann. Die Methoden, die jede einzelne Visualisierung voraussetzt, werden in den jeweiligen Kapiteln beschrieben.

Da eine Visualisierung aber auch möglichst interaktiv ist, gibt es über das Delegate Design Pattern [Apple, 2012b], das dem DataSource Pattern sehr ähnelt, die Möglichkeit

einem Delegate mitzuteilen, dass ein spezielles Event eingetreten ist. In der derzeitigen Implementierung des VisualizationKits haben alle Visualisierungen die Möglichkeit ihrem Delegate mitzuteilen, dass der Benutzer einen speziellen Datensatz hervorheben möchte.

Derzeit funktioniert dies bei allen zeitgebundenen Datensätzen, nicht allerdings bei kumulierten Datensätzen, wie sie beispielsweise in einem Tortendiagramm (siehe Kapitel 5.3.1 vorkommen. Es wäre allerdings recht einfach diese Funktionalität einzubauen.

Um als Visualisierung auf ein Hervorheben eines Zeitpunktes aufmerksam zu machen, reicht es folgende Delegate Methode aufzurufen:

```
1 @protocol VKVisualizationDelegate <NSObject>
2 - (void)visualization:(VKVisualization*)visualization
3  wantsToHighlightDate:(NSDate*)date;
4 @end
```

Listing 5.1: VKVisualizationDelegate

Eine gute Übersicht über alle implementierten Visualisierungen im VisualizationKit Framework bietet Abbildung 5.2.

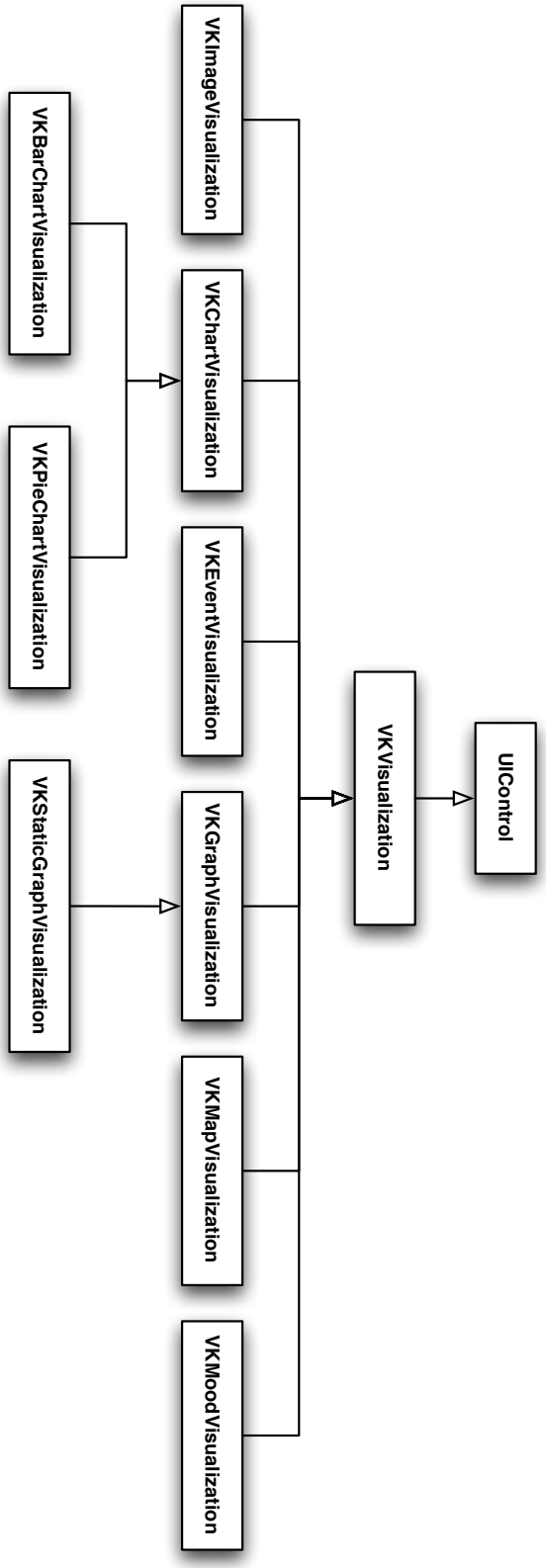

Abbildung 5.2: Klassen-Übersichts-Diagramm aller im VisualizationKit implementierten Visualisierungen

5.3 Visualisierungen

Im folgende Kapitel wird auf die einzelnen Visualisierungen eingegangen, die als Teil des VisualizationKit Frameworks implementiert wurden.

5.3.1 Balken und Tortendiagramm

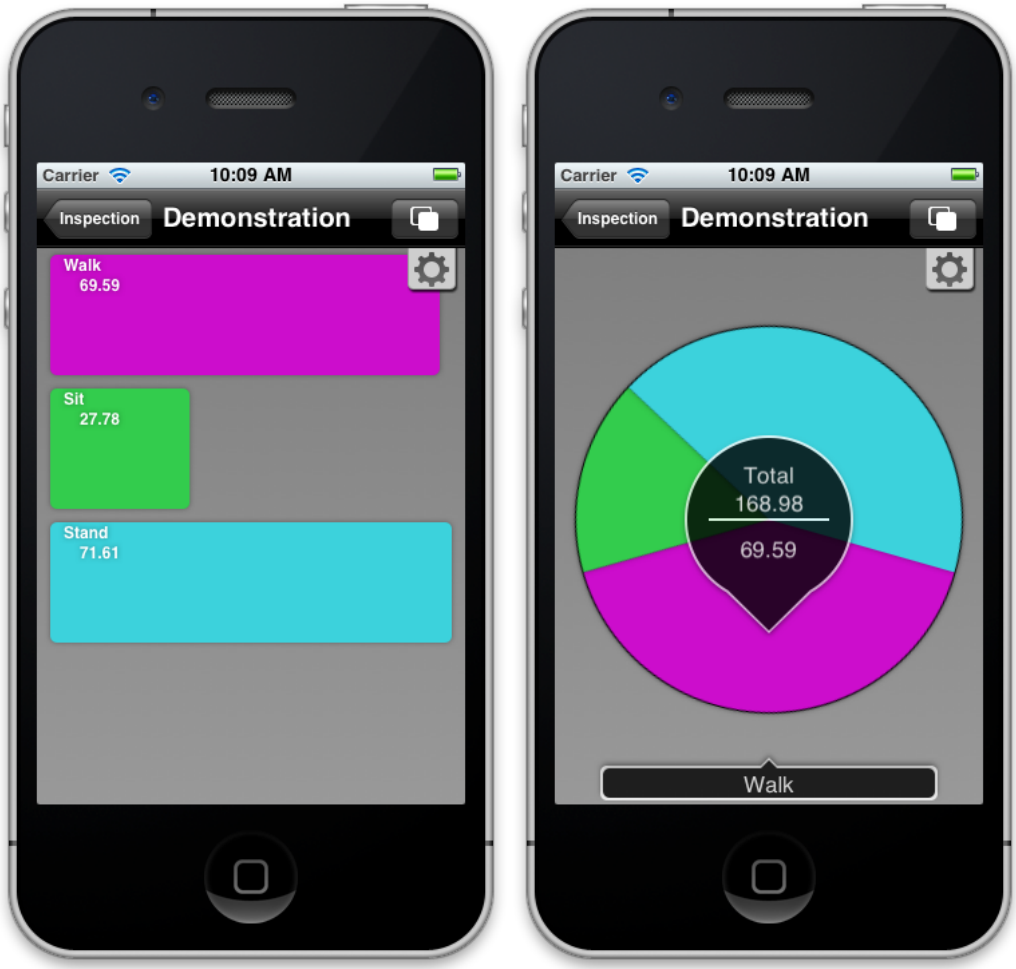

Abbildung 5.3: Balken und Tortendiagramm

Um absolute und prozentuelle Werte effizient darstellen zu können, wurden in dieser Masterarbeit zwei Chart Visualisierungen implementiert. Das ist zum Einen ein Balken Diagramm und zum Anderen ein Tortendiagramm. Verwendung findet diese Visualisierung in iPeeper, der im Zuge dieser Masterarbeit entstandene Prototyp (siehe Kapitel 6), für das Visualisieren von Bewegungsarten wie Gehen/Laufen/Liegen aber auch um Umgebungs-Lautstärkebereiche darzustellen.

Um beim Tortendiagramm, das selbst beinahe den gesamten Bildschirm einnimmt, Platz zu sparen, wurde auf eine eigene Legende verzichtet. Stattdessen ist das gesamte Diagramm drehbar. Wie in Abbildung 5.3 zu sehen, gibt es einen Pfeil der das aktuell selektierte Segment markiert, sowie die Anzeige des aktuellen Wertes in einem Textfeld in der Mitte des Diagramms. Ein weiterer Pfeil ist unter dem Tortendiagramm und ist ein weiteres Feld um den angezeigten Wert textuell anzuzeigen.

Zum Ansprechen von Torten- aber auch Balkendiagrammen gibt es wie für jede Visualisierung, eigene DataSource Methoden. Listing 5.2 zeigt die Methoden, die ein Controller implementieren muss, um die Toren- und Balkendiagramme darzustellen. Die Methoden sind für beide Diagramme gleich.

```
1 @protocol VKVisualizationDataSource <NSObject>
2 - (NSUInteger)numberOfSegementsInChart:(VKChart*)chart;
3 - (NSNumber*)valueForSegment:(NSUInteger)segment
4                       inChart:(VKChart*)chart;
5 @optional
6 - (NSString*)titleForSegmentWithIndex:(NSUInteger)segmentIndex
7                                 inChart:(VKChart*)chart;
8 @end
```

Listing 5.2: DataSource Methoden für Torten- und Balkendiagramme

5.3.2 Liniendiagramm

Um den Verlauf eines Wertes über die Zeit darstellen zu können, eignet sich ein Liniendiagramm (Abbildung 5.4) wohl am besten. Somit ist dieses auch als Teil des VisualizationKits implementiert worden.

Die größte Herausforderung bei der Darstellung von Liniendiagrammen war die sehr hohe Anzahl an zu visualisierenden Datenpunkten. So nimmt beispielsweise ein GPS Sensor, dessen Höhenwerte in einem Liniendiagramm schön zu visualisieren sind, bei einer einstündigen Autofahrt leicht über 6000 Datensätze auf. Der Soundsensor, der die Umgebungslautstärke misst, übersteigt diesen Wert recht leicht.

Die gesamte Information auf einen einzigen iPhone Bildschirm unterzubringen, wäre wohl sehr unübersichtlich. Aus diesem Grund lassen sich Liniendiagramme um die Zeitachse bewegen und auch zoomen um ausgewählte Teile höher aufzulösen.

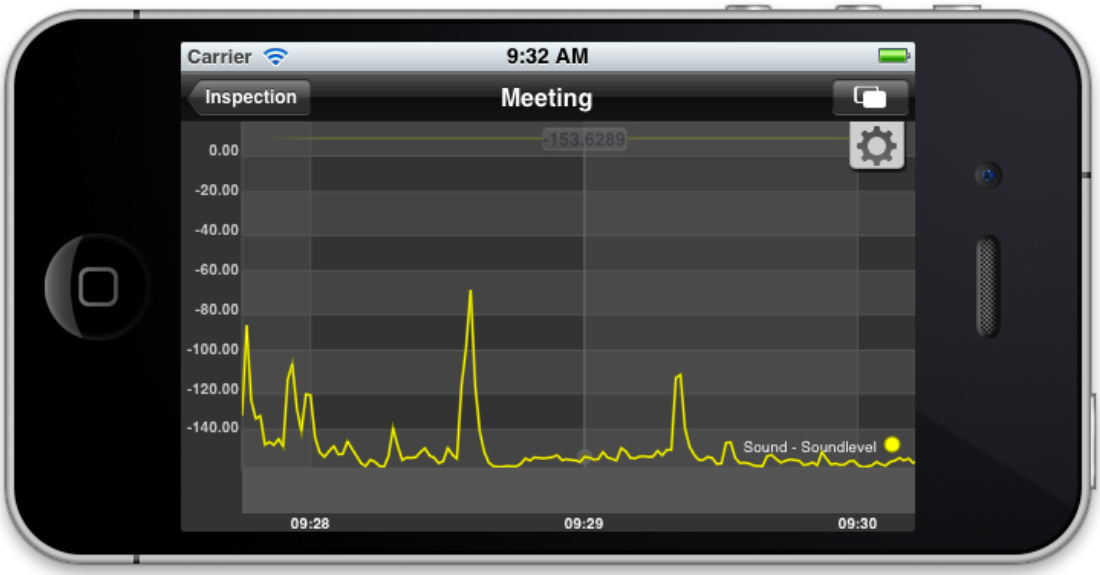

Abbildung 5.4: Liniendiagramm

Wie bereits in Kapitel 2.2.1 beschreiben, wäre das Vorrendern einer solchen Kurve sehr speicherintensiv. Aus diesem Grund speichert sich jeder Layer des Liniendiagramms (pro Layer wird eine Kurve angezeigt) nur einen Pfad mit den einzelnen Werten. Dieser wird dann immer wieder neu gezeichnet und obwohl sich alle Datensätze in einem solchen Pfad befinden, geht das selbst bei sehr vielen Punkten sehr schnell und flüssig. Ein weiterer Vorteil ist der, dass auch nach dem Zoomen die Linien immer noch komplett scharf dargestellt werden, was bei einer vorgerenderten Grafik nicht der Fall wäre. In dem Fall müsste man diese erst recht wieder neu zeichnen lassen, was den Hauptvorteil wieder zunichte machen würde.

Das DataSource Protokoll eines Liniendiagramms ist bereits ein bisschen umfangreicher als das der Balken- und Tortendiagramme. Listing 5.3 zeigt die benötigen Methoden.

```
1 @protocol VKVisualizationDataSource <NSObject>
2 - (NSUInteger)numberOfLinesInStaticGraph:(VKStaticGraph*)
    staticGraph;
3 - (NSUInteger)numberOfValuesForLineWithIndex:(NSUInteger)
    lineIndex
4                                 inStaticGraph:(VKStaticGraph*)
                                      staticGraph;
5 - (CGPoint)pointForValueWithIndex:(NSUInteger)pointIndex inLine
    :(NSUInteger)lineIndex
```

```
6                       inStaticGraph:(VKStaticGraph*)staticGraph;
7  - (CGFloat)maximumValueInStaticGraph:(VKStaticGraph*)staticGraph
       ;
8
9  @optional
10 - (NSString*)titleForLineWithIndex:(NSUInteger)lineIndex
11                        inStaticGraph:(VKStaticGraph*)staticGraph;
12 - (UIColor*)colorForLineWithIndex:(NSUInteger)lineIndex
13                   inStaticGraphView:(VKStaticGraph*)staticGraph;
14 - (CGFloat)witdhForLineWithIndex:(NSUInteger)lineIndex
15                  inStaticGraphView:(VKStaticGraph*)staticGraph;
16 - (CGFloat)minimumValueInStaticGraph:(VKStaticGraph*)staticGraph
       ;
17 - (CGFloat)startingOffsetInStaticGraph:(VKStaticGraph*)
       staticGraph;
18 - (CGFloat)intervalForVerticalGridLinesInStaticGraph:(
       VKStaticGraph*)staticGraph;
19 @end
```

Listing 5.3: DataSource Methoden für Liniendiagramme

5.3.3 Karte

Um GPS Sensordaten, also die Information über den geographischen Längengrad und Breitengrad darzustellen, eignet sich eine Karte sehr gut. Glücklicherweise stellt das Cocoa Touch Framework APIs für das Darstellen solcher Karten zur Verfügung.

Diese API erlaubt beispielsweise das Setzen von Stecknadeln, um einen Punkt auf recht einfachem Weg zu markieren. Um einen Pfad in die Karte zu zeichnen, ist ein wenig mehr Arbeit nötig. Glücklicherweise bietet die API aber Methoden, um eine Koordinate in einen Punkt einer UIView [Apple, 2012i] umzuwandeln.

```
1  - (CGPoint)convertCoordinate:(CLLocationCoordinate2D)coordinate
2                  toPointToView:(UIView*)view;
```

Durch diese Methode ist es recht einfach einen Pfad zu erzeugen und diesen in einem Layer über die Karte zu legen.

Um aber mehr Information zu liefern als nur den Pfad selbst, kodiert die im Zuge dieser

Abbildung 5.5: Kartenvisualisierung

Masterarbeit implementierte Karten Visualisierung auch die Information über die aktuelle Geschwindigkeit in Form von Farben entlang dieses Pfades. Wie in Abbildung 5.5 links oben zu sehen, ist eine Legende mit verschiedensten Geschwindigkeiten zu sehen. Somit bekommt man auf einen Blick sehr leicht noch die Information über die vorherrschende Geschwindigkeit mit.

Denkbar und ein schöne Erweiterung für die Kartenvisualisierung, wäre noch die Information über die geographische Höhe zu verpacken. Möglich wäre hier die Information beim Highlighten mittels der Abspielfunktion (siehe Kapitel 5.5) in einer Box anzuzeigen, oder auch über Farben zu kodieren. Man müsste hierbei aber speziell darauf achten, dass die Karte nicht zu unübersichtlich wird.

Eine weitere Erweiterung wäre das die Anzeige von Kompass Daten. Wie in Kapitel 2.1 beschrieben, verfügen moderne Smartphones über einen eingebauten Kompass. In Kombination mit der geographischen Positionsinformation könnte man somit auch die Blickrichtung des Benutzers feststellen. Voraussetzung dafür wäre allerdings, dass der Benutzer das Telefon auch immer in Blickrichtung hält.

Kapitel 2.1.7 beschreibt, dass es mit Bluetooth möglich ist eine ungefähre Anzahl an Personen, die sich in der Umgebung befinden, zu ermitteln. Die derzeitige Kartenvisualisierung ist dafür aber noch nicht ausgelegt, dies wäre aber auch eine schöne Erweiterungsmöglichkeit.

Da in dem VKMapViewLocation Objekt bereits alle Informationen, wie geographische Länge und Breite, aber auch Höhe und Geschwindigkeit gewrappt sind, ist das DataSource Protokoll für das Anzeigen von Karten Visualisierungen sehr schlank. In Listing 5.4 sind die beiden benötigten Methoden beschrieben.

```
1 @protocol VKVisualizationDataSource <NSObject>
2 - (NSUInteger)numberOfWaypointsForMapView:(VKMapView*)mapView;
3
4 - (VKMapViewLocation*)waypointAtIndex:(NSUInteger)index
5                               forMapView:(VKMapView*)mapView;
6 @end
```

Listing 5.4: DataSource Methoden für die Karten Visualisierung

5.3.4 Moodmap

Abbildung 5.6: Moodmap Visualisierung

Eine weitere Visualisierungen im VisualizationKit Framework ist die Moodmap [Sundström et al., 2007] [Russell, 1980]. Wenn ein Benutzer über seine Aufzeichnungen re-

flektieren will, sind von einem Sensor erfasste Werte schon mal ein recht guter Ansatzpunkt. Wie sich der Benutzer während dieser Aufzeichnung gefühlt hat, kann ein GPS-, Beschleunigungs- oder Lautstärke-Sensor allerdings nicht ermitteln. Deshalb bietet eine Moodmap in iPeeper, der Prototyp in dem das VisualizationKit Framework Einsatz findet (genauer beschrieben in Kapitel 6), die Möglichkeit, dem Benutzer direkt seine derzeitigen Gefühle in einer sogenannten Moodmap einzutragen zu lassen.

Die Moodmap, die als Teil des VisualizationKit Framework implementiert wurde, ist auf dem Circumplex Model von [Russell, 1980] basiert. Russell unterteilte eine quadratische Fläche, wie in Abbildung 5.7a zu sehen, in acht Bereiche, die kreisförmig angeordnet sind.

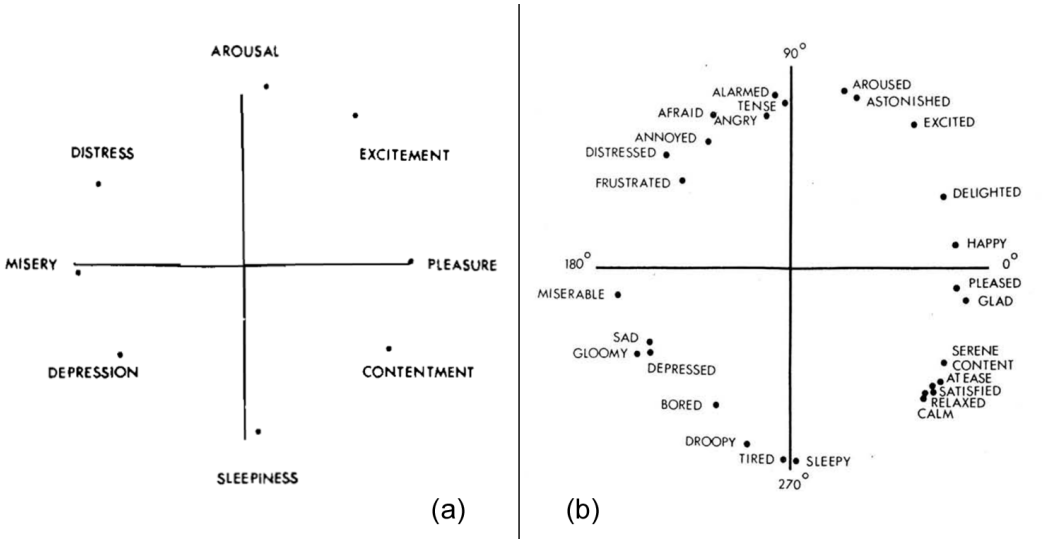

Abbildung 5.7: (a) Kreisförmige Unterteilung in 8 Bereiche zur Repräsentation von Gefühlen nach [Russell, 1980]
(b) Erweiterung dieser Darstellung in 28 Bereiche
Quelle: [Russell, 1980]

Diese Darstellung von Gefühlen nach dem Circumplex Model von Russell in diesem Koordinatensystem wurde nun noch um Farben erweitert. Diese Repräsentation basiert auf dem Farbensystem von [Itten, 2003] bzw. [Ståhl et al., 2005] und soll dem Benutzer helfen seine Gefühle in Form von Farben besser ausdrücken zu können.

Für die Auswertung der geklickten Punkte in der Moodmap kommt im VisualizationKit Framework, anders als in dem Teil, der in iPeeper aufzeichnet, noch eine Beschriftung der Achsen hinzu. Die X Achse ist demnach die “evaluation” Dimension, die mit “Feel good / Feel bad” beschriftet ist, die “activation” Dimension, die Y Achse, ist mit “High energy / Low energy” beschriftet.

Da das VisualizationKit Framework selbst keine Berechnungen bzw. Zuordnungen der Punkte macht, ist die Ansteuerung der Moodmap Visualisierung sehr einfach. Wie in Listing 5.5 zu sehen, gibt es nur eine Methode die zwei zwischen 0 und 1 normierte Werte, die in dem Objekt VKMood gewrappt sind, entgegen nimmt. Diese Werte entsprechen der "evaluation" bzw. der "activation".

```
1 @protocol VKVisualizationDataSource <NSObject>
2 - (VKMood*)moodClosestToDate:(NSDate*)date
3                 inVisualization:(VKVisualization*)visualization;
4 @end
```

Listing 5.5: DataSource Methode für die MoodMap

5.3.5 Bilder

Abbildung 5.8: Beispiel der Bild Visualisierung

Da ein Bild ja bekanntlich mehr als tausend Worte sagt, bietet das Visualization Kit Framework natürlich auch die Möglichkeit Bilder zu visualisieren. Technisch ist eine solche Visualisierung recht einfach umzusetzen. Das UIKit Framework von Apple [Apple, 2012g,

S 468] bietet eine UIImageView zur Darstellung von Bildern. Diese nimmt ein UIImage entgegen, dass wiederum mit einem einzigen API Aufruf, unter Übergabe des Dateipfades, geladen werden kann.

Mittels eines Sliders kann der Benutzer nun durch die einzelnen Aufnahmen blättern und so in der Zeit vor und zurück gehen. Vor allem in Kombination mit anderen Sensordaten, die man mittels des Synchronisations-Features (siehe Kapitel 5.4) synchronisiert werden, entsteht so ein schöner Rückblick von gespeicherten Ereignissen.

Wie in Kapitel 5.5 beschrieben, bietet das Visualization Kit Framework aber auch die Möglichkeit seine gespeicherten Daten wie einen Film abzuspielen. Da die Daten für Bilder, im Vergleich zu den Daten der anderen implementierten Visualisierung, vom Speicherbedarf her am größten sind, stellte sich hier die Frage, ob es performancetechnisch sinnvoll wäre, die Bilder in ein Video zu konvertieren. Bei Tests stellte sich allerdings heraus, dass es durchaus ausreichend ist, die Bilder on Demand nachzuladen und anzuzeigen, selbst wenn man seine Daten im schnellen Zeitraffer abspielt.

Da die Aufzeichnung von Bildern nicht unbedingt in einem fixen Intervall sein muss, ist für die Bildvisualisierung nicht einfach nur ein Zugriff per Index möglich. Da jede Visualisierung von ihrer Superklasse VKVisualization (Klassendiagramm in Abbildung 5.1) ein Start- und Enddatum Property erbt, kennt die Bildvisualisierung seinen Darstellungsbereich und kann somit jede Position des Sliders in der Visualisierung direkt in ein Datum umgerechnet werden. Somit reicht nun die DataSource Methode zum Darstellen von Bildern in der Bildvisualisierung.

```
1 @protocol VKVisualizationDataSource <NSObject>
2 - (UIImage*)imageClosestToDate:(NSDate*)date
3                 inVisualization:(VKVisualization*)visualization;
4 @end
```

Listing 5.6: DataSource Methode für die Darstellung von Bildern

5.3.6 Events

Da mobile Geräte nicht nur Daten zur Verfügung stellen ,die in den bisher vorgestellten Visualisierungen darstellbar sind, bietet das Visualization Kit Framework, wie in Abbildung 5.9 zu sehen, eine Möglichkeit textbasierte Events darzustellen. Denkbare Datensätze für eine solche Text-basierte Visualisierung sind beispielsweise Emails oder SMS über die ein

Abbildung 5.9: Beispiel der Event-Visualisierung, die ausgewertete Moodmap Daten präsentiert

Benutzer gerne reflektieren würde, da diese einen wesentlichen Einfluss auf Reaktionen und Gefühle haben können. Es sind aber auch Nachrichten, die man beispielsweise per RSS geliefert bekommt, denkbare Informationen für die Event-Visualisierung.

[Shilton, 2009, S. 5] beschreibt weiters, dass es durchaus interessant ist, dem Benutzer Alternativen zu bekannten Darstellungsvarianten zu geben. Schaut man sich beispielsweise die Moodmap Visualisierung an, so ist das Interpretieren der ermittelten Werte unter Umständen in Textform einfacher. Somit würde der Benutzer beispielsweise nicht den gedrückten Punkt auf der farbigen Moodmap sehen, sondern in der Event-Visualisierung den klassifizierten Text-basierten Wert sehen. Für eine Kartendarstellung wäre es beispielsweise auch denkbar statt einen Punkt auf einer Karte angezeigt zu bekommen, zu lesen, dass man zu einem bestimmten Zeitpunkt im Büro angekommen ist.

Wie man sieht, können die Inputs in die Event-Visualisierung recht vielfältig sein. Deshalb ist auch das Protokoll für den DataSource, wie in Listing 5.7 zu sehen, etwas länger:

```
1 @protocol VKVisualizationDataSource <NSObject>
2 - (NSUInteger)numberOfEventsInVisualization:(VKVisualization*)
    visualization;
3 - (NSDate*)timestampOfEventWithIndex:(NSUInteger)eventIndex
```

```
4                          inVisualization:(VKVisualization*)
                              visualization;
5  - (NSString*)textOfEventWithIndex:(NSUInteger)eventIndex
6                       inVisualization:(VKVisualization*)
                           visualization;
7  @optional
8  - (NSString*)authorOfEventWithIndex:(NSUInteger)eventIndex
9                         inVisualization:(VKVisualization*)
                             visualization;
10 - (UIImage*)previewThumbOfEventWithIndex:(NSUInteger)eventIndex
11                           inVisualization:(VKVisualization*)
                                visualization;
12 - (VKEventType)eventTypeOfEventWithIndex:(NSUInteger)eventIndex
13                           inVisualization:(VKVisualization*)
                                visualization;
14 @end
```

Listing 5.7: DataSource Methoden für Events

5.4 Synchronisation von aufgezeichneten Daten

Da in dieser Masterarbeit, wie bereits beschrieben, eine generische Methode der Datendarstellung herangezogen wurde, die sich zur Darstellung von vielen unterschiedlichen Daten eignet, anstatt eine maßgeschneiderte Lösung zu implementieren, war die größte Herausforderung einen Konnex zwischen den einzelnen Visualisierungen zu schaffen. Der Vorteil von speziell zugeschnittenen Darstellungsvarianten, wie sie zum Beispiel im Affective Diary (siehe Abbildung 4.8) vorkommen, ist der, dass man die ganze Information auf einen Blick sehen kann. Eine solche Visualisierungsvariante ist aber nicht leicht erweiterbar und vor allem auf ein fixes Set an Inputs abgestimmt. Um nun das beste aus beiden Welten zu erhalten, wurde im VisualizationKit eine Lösung erarbeitet, die es erlaubt auch mit herkömmlichen Charting Methoden möglichst viel Information und vor allem der Zusammenhang zwischen den einzelnen Datensätzen zu veranschaulichen.

Das Hauptproblem, wenn man beispielsweise zwei Liniendiagramme nebeneinander sieht, ist dass man sich unter Umständen recht schwer tut zu sehen, welcher Wert mit einem anderen gleichzusetzen ist. Eine mögliche Problemlösung wäre es hier, einfach mehrere Kurven in einem Liniendiagramm darzustellen. Wenn dann eine der beiden Kurven aller-

dings einen völlig anderen Wertebereich hat, wie in dem Beispiel in Abbildung 5.10 zu sehen, kann das unter Umständen zu noch größerer Unübersichtlichkeit führen.

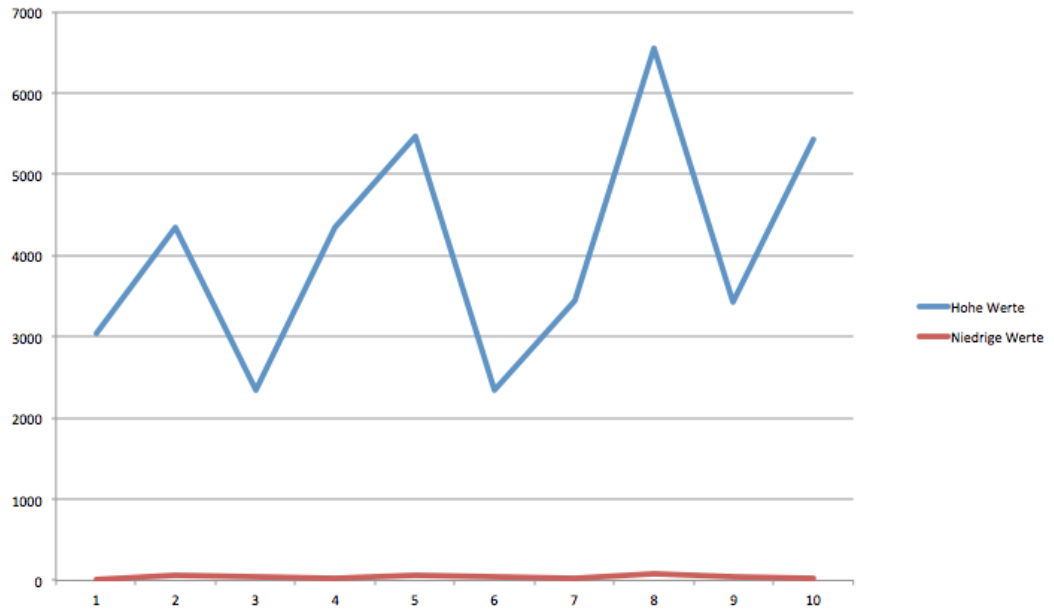

Abbildung 5.10: Die Werte der roten Kurve sind schwer zu erkennen, da sich der Wertebereich stark von der blauen Kurve unterscheidet

Eine Lösung wäre es eine zweite Werte-Achse einzuführen, was allerdings wieder zu Unübersichtlichkeit führen würde, da sich der Betrachter eventuell schwer tun würde, die jeweilige Kurve der richtigen Beschriftungsachse zuzuordnen.

Wenn man jetzt einfach zwei getrennte Kurven betrachten würde, hat man wiederum das Problem einen Wert einer Kurve, einem Wert einer anderen Kurve gleichzusetzen, da man zuerst den entsprechenden Bereich finden müsste.

Noch größer wird das Problem wenn man sich Daten anschauen will, die nicht mit der gleichen Darstellungsmethode visualisiert werden können. Beispielsweise wenn man sich die Bilder, die die Kamera aufgezeichnet hat, mit deren Geo-Position vergleichen will. Die Bilder wären jetzt noch als kleines Overlay in einer Kartenvisualisierung denkbar. Will man jetzt aber vielleicht noch die Höhenkurve dieser Strecke sehen und dann vielleicht auch noch die vorherrschende Lautstärke, welche komplett unterschiedliche Einheiten und somit Wertebereiche haben, so wird das in einer einzigen Visualisierung unter Umständen sehr unübersichtlich.

Aus diesem Grund war das Ziel der praktischen Implementierung dieser Masterarbeit, ein Visualisierungs-Framework zu schaffen, mit dem man mobile Sensordaten einfach vi-

sualisieren kann und diese vor allem interaktiv miteinander in Zusammenhang bringen kann. Da alle Daten wie bereits in Kapitel 3 erwähnt, über einen gemeinsamen Messzeitpunkt verfügen, ist es möglich die Daten über diesen zu synchronisieren. Abbildung 5.11 zeigt wie diese Synchronisation am iPad aussieht. Der Benutzer bleibt mit seinem Finger länger auf einer Stelle der Visualisierung über die er gerne mehr Informationen hätte. Daraufhin wird der gleiche Zeitpunkt auch in anderen Visualisierungen gehighlightet. Dabei ist es egal was die einzelnen Visualisierungen darstellen, sofern es sich um Daten handelt, die einem genauen Zeitpunkt zugeordnet werden können. In dem Beispiel aus der Abbildung sieht man das vorher angesprochene Szenario mit Bildern, Geo-Position, Höhenverlauf sowie der vorherrschenden Lautstärke.

Abbildung 5.11: Synchronisation der Visualisierungen von Höheninformationen, Lautstärke-Levels, Bild und Geo-Positionsdaten am iPad

Technisch wird dies über das Delegate/Datasource Design Pattern umgesetzt [Apple, 2012b]. Nachdem der Benutzer den Finger länger auf einem Punkt gedrückt hält, wird das von einem UILongPressGestureRecognizer [Apple, 2012h] erkannt. Daraufhin kann die Visualisierung mittels der Delegate Methode

```
1 - (void)visualization:(VKVisualization*)visualization
2   wantsToHighlightDate:(NSDate*)date;
```

ihrem VisualizationController mitteilen, dass sie ein gewisses Datum highlighten will. Dieser wiederum kann über die Instanz Methode,

```
1 - (void)highlightDate:(NSDate*)date;
```

die jede Visualisierung von der abstrakten Superklasse VKViszualization erbt, jeder seiner Visualisierungen mitteilen, dass diese das übergebene Datum markieren sollen.

5.5 Abspielen von aufgezeichneten Daten

Um es dem Benutzer noch leichter zu machen über seine Aufzeichnungen reflektieren zu können [Boud, 1985], bietet das VisualizationKit Framework die Möglichkeit, Visualisierungen wie in einem Film abzuspielen. Dazu gibt es wie in Abbildung 5.11 zu sehen, am unteren Rand die dafür gängigen Buttons “An den Anfang springen”, “langsamer Abspielen”, “Play/Pause”, “schneller Abspielen” und “Zum Ende springen”. Ein Klick auf den Play Button instanziiert einen Timer, der in einem Intervall, das sich anpassen lässt, die Methode

```
1 - (void)highlightDate:(NSDate*)date;
```

aufruft. Das ist die gleiche Methode, die sich schon die Synchronisation der Daten, die in Kapitel 5.4 beschrieben wurde, zu Nutze gemacht hat.

Auf diese Art und Weise ist es dem Benutzer möglich ohne weitere Interaktion die aufgezeichneten Daten in chronologisch korrekter Reihenfolge Revue passieren zu lassen.

Kapitel 6

Prototyp - iPeeper

"The eye sees only what the mind is prepared to comprehend." *Davies Robertson*

Im Zuge dieser Masterarbeit ist ein Prototyp namens "iPeeper" entstanden. Dieser ist Ergebnis aus den Masterarbeiten von [Edler, 2012], der sich mit der Erfassung und dem Verarbeiten von mobilen Sensor Daten befasst hat, und mir. Dieser Prototyp wird in diesem Kapitel vorgestellt.

Der Grund für diesen gemeinsamen Prototypen war der, dass eine Visualisierung Daten benötigt um etwas anzeigen zu können und umgekehrt Daten eine Visualisierung benötigen, um besser und leichter verständlich dargestellt zu werden. Hauptbestandteile von iPeeper sind somit das von [Edler, 2012] entwickelte *SensorKit (SK)* und *SensorModel (SM)* Framework, sowie das von mir entwickelte *VisualizationKit (VK)* Framework.

Da es sich um einen gemeinsamen Prototypen handelt, ist auch dieses Kapitel in Zusammenarbeit entstanden und kommt somit in beiden schriftlichen Masterarbeiten wortgleich vor.

6.1 Übersicht

Bei iPeeper handelt es sich um eine iPhone/iPad Applikation um User-Context Daten aufzuzeichnen und zu visualisieren. In Kapitel 6.2 wird genauer auf die Datenaufzeichnung eingegangen und Kapitel 6.3 behandelt die Datenvisualisierung.

Das Poster, das in Abbildung 6.1 zu sehen ist, wurde für iPeeper im Zuge der i-KNOW

2011[1] erstellt und auf der Konferenz dem Publikum präsentiert.

Abbildung 6.1: Poster für iPeeper, dass auf der i-KNOW'11 Konferenz präsentiert wurde

Als wir mit dem Erstellen von iPeeper begannen, war es uns wichtig einen Prototypen zu erstellen, der auf beiden iOS Plattformen (iPhone sowie iPad) gut zu benutzen ist. Später hat sich allerdings herausgestellt, dass sich das iPhone auf Grund seiner kleineren Größe und seiner komfortableren Möglichkeit es zu transportieren, besser als Aufzeichnungsgerät eignet. Das iPad hingegen ist dafür prädestiniert um darauf die aufgezeichneten Daten auf dem größeren Display analysieren zu können. Beides, das Aufzeichnen sowie das Visualisieren, ist aber prinzipiell mit beiden Geräten möglich.

[1] i-KNOW 2011, 11th International Conference on Knowledge Management and Knowledge Technologies, 7-9 September 2011, Messe Congress Graz, Austria, http://i-know.tugraz.at/ (zuletzt besucht am 28. April 2012)

Da aber nun der Datenaustausch zwischen den Geräten ein wichtiges Thema wurde, wurde ein weiterer Bestandteil, neben dem SensorKit, SensorModel und VisualizationKit Framework eingebaut. Dabei handelt es sich um eine Schnittstelle zum Austausch von Daten zwischen den Geräten. Mehr dazu folgt in Kapitel 6.4.

6.2 Datenaufzeichnung

Bevor wir vom Versenden und Visualisieren der Daten sprechen können, müssen wir das Aufzeichnen dieser Daten behandeln. In iPeeper wurde das Konzept von sogenannten "Sessions" eingeführt (siehe Kapitel 6.5). Wann auch immer ein Benutzer Daten aufzeichnen will, muss er diese Daten einer zeitlich abgegrenzten und geschlossenen Einheit, einer Session, zuordnen. Das kann beispielsweise ein Meeting, eine Zugfahrt oder ein Einsatz sein.

Abbildung 6.2: Anlegen einer neuen Session

Wie in Abbildung 6.2 zu sehen ist, muss man für jede neue Session zuerst einen Namen vergeben. Das kann der Benutzer entweder manuell machen, oder man kann sich auch

einen der vorgeschlagenen Namen aussuchen. Derzeit sind das noch statische Texte, für zukünftige Erweiterungen ist es aber geplant auch aktuelle Ereignisse aus dem Kalender des Benutzers auszulesen und vorzuschlagen bzw. zuletzt verwendete Session Namen zu verwenden.

Nachdem man seiner Aufzeichnung einen Namen gegeben hat, kommt der Benutzer weiter zur zweiten und bereits letzten Einstellung vor dem echten Aufzeichnen. Wie in Abbildung 6.3 zu erkennen, sucht man sich hier die Sensoren aus, die Daten erheben sollen. Einzelne Sensoren haben auch die Möglichkeit individuelle Einstellungen an ihnen vorzunehmen. Aktuell trifft das nur auf den Kamera Sensor zu. Diese Einstellungen sind über das blaue Pfeil Icon neben dem Sensornamen zu erreichen. Im Falle der Kamera sind das beispielsweise die Bildqualität und das Intervall, in dem Bilder gemacht werden sollen.

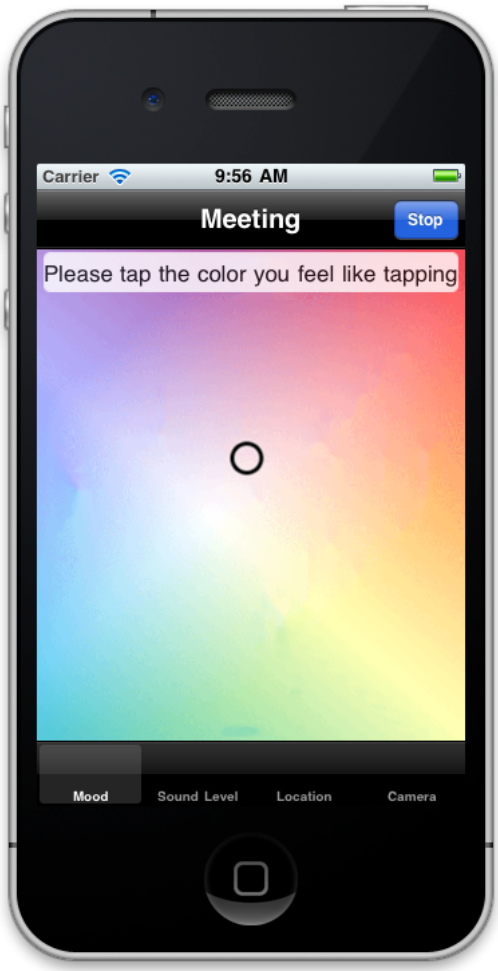

Abbildung 6.3: Aufzeichnen einer neuen Session

Nachdem nun neben dem Session Namen auch die genutzten Sensoren ausgewählt wur-

den, kann man mit dem tatsächlichen Aufzeichnen starten. Wie in Abbildung 6.3 zu sehen, können während des Aufzeichnens durchaus auch weitere nicht ausgewählte Sensoren aktiv sein. Das sind sogenannte passive Sensoren. Passive Sensoren sind zwar immer aktiv, zeichnen aber nur durch direkten User-Input Daten auf. In dem Fall des Screenshots ist das eine Moodmap (siehe Kapitel 5.3.4).

Für die Zukunft sind weitere passive Sensoren geplant, die beispielsweise das Eintragen einer Notiz ermöglichen. Weitere Details dazu sind in der Masterarbeit von [Edler, 2012] beschrieben.

Wie man durch die Tabs am unteren Bildschirmrand in Abbildung 6.3 sehen kann, gibt es neben den passiven auch aktive Sensoren, die ihren aktuellen Wert anzeigen.

6.3 Datenvisualisierung

Nachdem nun eine Session aufgezeichnet wurde, hat der Benutzer die Möglichkeit, sich diese visualisieren zu lassen. Zuerst muss man dazu die entsprechende Session auswählen. Um diesen Vorgang zu erleichtern, kann der Benutzer über den Kalender auf den jeweiligen Tag wechseln, an dem die Session aufgezeichnet wurde.

Sofern an dem ausgewählten Tag eine Aufzeichnung stattgefunden hat, ist diese (wie in Abbildung 6.4 ersichtlich) in der Liste neben bzw. unter dem Kalender sichtbar. Für jede Art von aufgezeichneten Daten gibt es nun verschiedenste Visualisierungen. Da sich aber nicht alle Visualisierungen für alle Daten eignen, werden auch immer nur die Visualisierungen angezeigt, die mit den ausgewählten Daten möglich sind. Am iPad ist es möglich bis zu vier Visualisierungen gleichzeitig darzustellen. Um das Layout zu ändern, reicht ein Klick auf den Button links unten, der in Abbildung 6.5 mit “1” markiert ist.

Da der Platz am iPhone nicht so groß ist, kann immer nur eine Visualisierung auf dem Bildschirm angezeigt werden, visualisiert werden aber mehr. Ein Umschalten ist am iPhone dann über den Button rechts oben mit der Beschriftung “2” möglich.

Wie in Abbildung 6.6 zu sehen ist, ist es im nächsten Schritt möglich, passende Datensätze auszuwählen, die von der zuvor festgelegten Visualisierung dargestellt werden können. Dabei ist es je nach Visualisierung möglich auch mehrere Datensätze auszuwählen. So kann beispielsweise die Liniendiagramm Visualisierung mehrere Linien zeichnen, oder ein Tortendiagramm Daten aus mehreren Datensätzen zusammenfügen.

Das war auch der Grund die Menüführung der letzten beiden Schritte in dieser Reihen-

Abbildung 6.4: Auswahl einer Session für die Visualisierung derer Daten

folge festzulegen. Es wäre sonst möglich gewesen mehrere Datensätze zu wählen, die in der Kombination von keiner Visualisierung dargestellt werden könnten. Ein Beispiel wäre hier, wenn der Benutzer Daten vom Bewegungssensor (dessen Werte Stati wie “gehen”, “liegen”, “sitzen” sein können) mit den Daten vom Bildsensor und vom GPS Modul mischt. Dazu bräuchte man eine Balken- oder Torten-Diagramm Visualisierung für den Bewegungssensor, eine Karte für die GPS Positionen und noch eine Bildvisualisierung für die Bilder. Natürlich wäre es möglich gewesen eine Visualisierung zu erstellen, die all diese Informationen darstellen kann, doch wie bereits in Kapitel 3 beschrieben, war es die Entscheidung für diese Masterarbeit generische Visualisierungen zu erstellen.

Nachdem die Visualisierung und die dazu passenden Datensätze gewählt sind, können die Visualisierungen angezeigt werden. Das kann beispielsweise wie in Abbildung 6.7 aussehen. Hier sieht man die Visualisierung von Lautstärke (links oben), GPS Höheninformation (rechts oben), GPS Daten in einer Karte (links unten), sowie das dazugehörige Bild (rechts unten). Dabei sind die Daten mittels drücken und halten synchronisiert. Weitere Informationen zum Synchronisieren sind in Kapitel 5.4 beschrieben.

Abbildung 6.5: Auswahl der verfügbaren Visualisierungen

6.4 Datenaustausch

Beim Datenaustausch zwischen iPhone und iPad wäre es am einfachsten gewesen, das von Apple entwickelte GameKit Apple [2012c] zu verwenden. GameKit ermöglicht das Finden von anderen Geräten die sich im gleichen WLAN bzw. in Bluetooth Reichweite befinden, mittels einer schön abstrahierten API und bereits bestehendem User-Interface. Weiters gibt es für das Paaren von Geräten und dem Übertragen von Daten auch einfach zu verwendende Methoden.

Der Grund warum für den Prototypen allerdings eine eigene Lösung implementiert wurde ist der, dass es möglich sein sollte auch Daten mit dem KnowSe Framework [Rath, 2010], das auch am Know-Center [2] im Zuge eine Doktorarbeit entwickelt wurde, auszutauschen. Des weiteren ist eine Portierung von iPeeper auf andere mobile Plattformen wie zum Beispiel das Android Betriebssystem, für die Zukunft geplant und somit war es wichtig einen offenen Standard zu verwenden.

[2] http://know-center.tugraz.at/ueber-uns (zuletzt besucht am 12. August 2011)

Abbildung 6.6: Auswahl der verfügbaren Daten

Trotzdem wäre es schade gewesen den Vorteil des automatischen Erkennens von anderen, im gleichen Netzwerk befindlichen Geräten zu verlieren. Deshalb wurde vor dem tatsächlichen Datenaustausch noch eine weitere Schicht eingeführt, die nur dem Finden von verfügbaren anderen mobilen Geräten, die auch den iPeeper Service laufen haben, dient. Um das möglichst einfach zu realisieren wurde auf den Bonjour [Apple, 2012a] Dienst von Apple zurückgegriffen. Dieser steht auch für viele andere Plattformen (darunter Mac OS X, Windows und Android) zur Verfügung.

Die Lösung, die in iPeeper verwendet wird, ist somit eine hybride Lösung, die eine Netzwerkkommunikation über Sockets verwendet und für das Finden von anderen Geräten auf Bonjour (siehe Kapitel 6.4.1) zurückgreift.

Die genaue Spezifikation des Datenformates wird in Kapitel 6.4.2 beschrieben.

6.4.1 Finden von anderen iPeeper Instanzen

Bei der in iPeeper implementierten Lösung zum Datenaustausch kommt das soeben angesprochene Bonjour [Apple, 2012a] zum Einsatz, um das Finden anderer Geräte in der

Abbildung 6.7: Anzeige der aufgezeichneten Daten in iPeeper

Umgebung zu erleichtern. Bei Bonjour handelt es sich um ein von Apple entwickeltes *Zero-Configuration* Netzwerk Protokoll zum Finden und Verbinden von Diensten, die im lokalen aber auch globalen Netzwerk angeboten werden. iPeeper beschränkt sich allerdings auf das lokale Netzwerk.

Beim Start des Prototypen iPeeper wird zu Beginn ein Bonjour Service mit dem Namen “_knowse._tcp."gestartet. Andere im Netzwerk befindliche Instanzen von iPeeper merken das und listen die neue Instanz als verfügbaren Austausch Partner auf (siehe Abbildung 6.9). Wenn sich jemand dazu entscheidet eine aufgezeichnete Session mit jemand anderen zu teilen, muss zuerst dieses andere Gerät gefunden werden. Nachdem das Gerät ausgewählt wurde und man die Sensordaten, die man teilen will, ausgesucht hat, kann man mit dem Klick auf “Send Session"die Daten übertragen. Um die Daten dann tatsächlich zu senden, wird eine neue TCP/IP Socket Verbindung aufgemacht, über die dann kommuniziert werden kann. Den genauen Ablauf dieser Kommunikation ist im nächsten Abschnitt beschrieben.

Abbildung 6.8: Logo des Bonjour Services, Quelle: [Apple, 2012a]

Abbildung 6.9: Versenden einer aufgezeichneten Session

6.4.2 Protokoll

Wie bereits beschrieben, wurde beim Datenaustausch von iPeeper darauf geachtet, auf das proprietäre GameKit für die Übertragung der aufgezeichneten Sensordaten zu verzichten und eine eigene Lösung mittels Sockets implementiert. Nachdem eine Verbindung erfolgreich zu Stande gekommen ist, können Daten ausgetauscht werden. Das Format in dem Pakete übertragen werden ist in Abbildung 6.10 beschrieben.

Jedes Paket kann beliebig groß sein. Damit der Empfänger jedoch weiß wann er aufhören kann auf weitere Daten zu warten und damit beginnen kann ein Datenpaket zu entpacken, geben die ersten 4 Byte jeweils die Länge des kompletten Paketes an. Danach kommt ein Byte, der über die Kodierung der Daten Auskunft gibt. Derzeit versteht iPeeper folgende Typen:

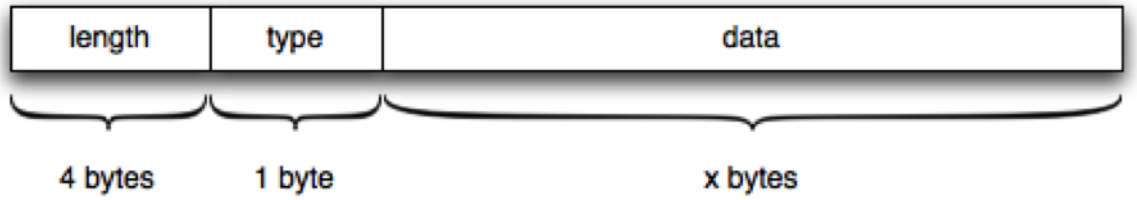

Abbildung 6.10: Aufbau von Netzwerk Paketen für den Datenaustausch in iPeeper

Types	
0x01	UTF8 konformer JSON String
0x02	Base64 kodiertes Raw Paket

Diese Kodierung ist für den Empfänger wichtig um zu wissen was er mit den erhaltenen Bytes anfangen soll. Der Großteil der Kommunikation wird mittels JSON Paketen abgewickelt. Auch die meisten Sensordaten werden in JSON formatierten Strings übertragen. Da es aber auch möglich ist "größere" Daten, wie beispielsweise Bilder, zu übertragen, werden diese im JSON nur als Referenz angegeben. Der Empfänger kann diese Referenzen dann getrennt anfordern und diese werden dann Base64 kodiert übertragen. Diese Aufteilung der Übertragung ist deshalb entstanden, um mobilen Geräten, die begrenzte Hardwareressourcen haben, die Möglichkeit zu geben, Bilder einzeln anzufordern, auf ihre Festplatte zu schreiben und den Speicher der zum Empfangen nötig ist, möglichst schnell wieder freizugeben.

Bei der Kommunikation werden grundsätzlich zwei unterschiedliche Szenarien unterschieden. Zum einen kann ein Gerät von sich aus Daten senden und zum anderen, kann es sein, dass ein Gerät die Daten eines anderen abrufen will. In der derzeitigen Implementierung von iPeeper wird allerdings nur auf den ersten Fall Rücksicht genommen. Es war aber denkbar, dass das KnowSe Framework des Know-Center [3] aber in Zukunft auch von sich aus um Daten bitten kann, weshalb auch dieser Use-Case bei der Konzeption des Datenaustausches berücksichtigt wurde.

Das Netzwerk-Ablauf-Diagramm in Abbildung 6.11 zeigt dabei den Fall, dass eine Instanz von iPeeper von sich aus Daten verschicken will, Abbildung 6.12 zeigt den gerade beschrieben Fall, in dem Daten von jemanden angefordert werden.

[3] http://know-center.tugraz.at/ueber-uns (zuletzt besucht am 12. August 2011)

6.5 Privatsphäre

Ein wichtiger Punkt beim Austausch von privaten Sensordaten ist die Privatsphäre. Da die aufgezeichneten Daten, die von den bereits bestehenden Sensoren, viel mehr aber noch Sensoren die in der Zukunft implementiert werden sollen (siehe Kapitel 6.6), oft sehr privat sein können, war das Thema Privatsphäre sehr wichtig.

Aus diesem Grund wurde zum einen das Konzept von Sessions entwickelt. In der ersten Version des Prototypen iPeeper wurden direkt nach dem Start immer alle Daten die irgendwie aufgezeichnet werden konnten, auch tatsächlich aufgezeichnet. Das hatte zum Nachteil, dass man auf der Suche nach einem speziellen Ereignis, zum Beispiel einem Meeting das man mitprotokolliert hat, um es jemand anderen zu zeigen, beinahe unausweichlich auch andere Daten sehen konnte, die in der zeitlichen Umgebung des Meetings erfasst wurden. Da diese Daten durchaus sensibel sein könnten, war es wichtig einen Mechanismus zu entwerfen, der einer solchen Situation vorbeugt. Weiters war eine weitere Anforderung an iPeeper die Möglichkeit Daten mit anderen Geräten auszutauschen. Auch für diese Funktion war das ungekapselte Aufnehmen aller Daten hinderlich.

Deshalb gibt es in iPeeper nun das Konzept von “Sessions”. Eine Session ist ein zeitlich abgegrenztes Ereignis, das mit einem Namen versehen wird. Dieser Name kann selbstständig gewählt werden. Geplant ist allerdings, dass iPeeper in Zukunft anhand vom Terminkalender auch Namen für die neu angelegte Session vorschlägt. Zu jeder Session kann der Benutzer in einem weiteren Schritt selbst bestimmen, welche Sensoren Daten aufzeichnen sollen. Seit es das Konzept von Sessions in iPeeper gibt, ist es nun einfacher gezielt an ein Set von Daten zu kommen.

Neben dem Konzept von Sessions, die Daten schon in kleinere Pakete unterteilen, war es aber auch beim Versenden von Daten wichtig, auch hier noch einmal filtern zu können. Wenn ein Benutzer beispielsweise ein Meeting mitprotokolliert, will er eventuell nach dem Meeting mit einem Kollegen darüber reflektieren [Boud, 1985]. Wenn diese Meeting-Session allerdings sehr private Daten enthält, wie beispielsweise seine eigenen Gefühle, die mittels der Moodmap (siehe Kapitel 5.3.4) eingegeben wurden, so will man diese eventuell nicht teilen, die anderen objektiveren Daten aber schon. Aus diesem Grund ist es in iPeeper beim Versenden von Daten möglich, zu filtern, welche Daten tatsächlich aus der ausgewählten Session übertragen werden.

Diese beiden Mechanismen zur Unterstützung der Privatsphäre sind natürlich nicht aus-

reichend um alle sensiblen Daten zu schützen, aber zumindest ein erster Ansatz. Weitere Ideen zur besseren Sicherung der Privatsphäre werden im nächsten Kapitel vorgestellt.

6.6 Zukünftige Arbeiten an iPeeper

Für die Zukunft ist es geplant, den in dieser Masterarbeit entstandenen Prototypen iPeeper, weiter zu entwickeln. Dabei wäre ein erster Schritt weitere Sensoren anzubieten. Das derzeitige SensorKit [Edler, 2012] bietet die Möglichkeit von iOS Geräten direkt angebotene Sensoren anzusprechen und auszulesen. Noch ist keine Möglichkeit vorgesehen auch externe Sensoren, die sich beispielsweise über Bluetooth paaren ließen, zu integrieren. Interessant wäre es hier mehr gesundheitsrelevante Informationen über den Benutzer zu erhalten, wie beispielsweise den Puls, Blutdruck oder ähnlichen.

Die möglichen Informationen die eine iOS Applikation auf Grund ihrer Umgebung in einer Sandbox [Apple, 2012d, S. 131], in der sie aus Sicherheitsgründen leben muss, beziehen kann, sind leider recht limitiert. So kann iPeeper, sobald es in den Hintergrund gelegt wird, keine Daten mehr aufzeichnen, da Aktivitäten von Applikationen in diesem Zustand stark limitiert sind. Auch hat man beispielsweise wenig Zugriff auf privatere Daten des Benutzers, die natürlich nur nach expliziter Erlaubnis, sicher spannende neue Schlüsse ziehen lassen würden.

Ein weiterer Punkt an dem man zukünftig noch an iPeeper Erweiterungen vornehmen könnte, sind Privatsphären Optimierungen, wie sie schon kurz im vorherigen Kapitel angesprochen wurden. So wäre beispielsweise eine Verschlüsselung der am Gerät abgespeicherten Daten wünschenswert, sowie ein Schutz der kompletten Applikation über eine Login-Möglichkeit.

Beim Versenden von Benutzer-Context Daten die mittels iPeeper aufgezeichnet wurden, wäre eine noch fein granularere Auswahl der zu versendenden Datensätze wünschenswert. Derzeit passiert die Übertragung an sich auch noch unverschlüsselt und in Plain-Text, was natürlich auch ein guter Ansatzpunkt wäre, um weitere Arbeiten an iPeeper durchzuführen.

Wie in Kapitel 5.4 beschrieben wurde, bietet das VisualizationKit Framework, das in iPeeper Daten visualisiert, die Möglichkeit zeitpunktbezogene Informationen zu synchronisieren. Schön wäre es allerdings, wenn es auch möglich wäre Zeiträume zu markieren. So könnte dann zum Beispiel eine Balkendiagramm Visualisierung die entsprechenden Zeiträume, in denen die verantwortlichen Messungen durchgeführt wurden, in einem Liniendia-

gramm hervorheben. Umgekehrt wäre es schön, eine passende Säule in einem Balkendiagramm hervorzuheben, wenn in einem Liniendiagramm ein Zeitpunkt zur Synchronisation ausgewählt wurde.

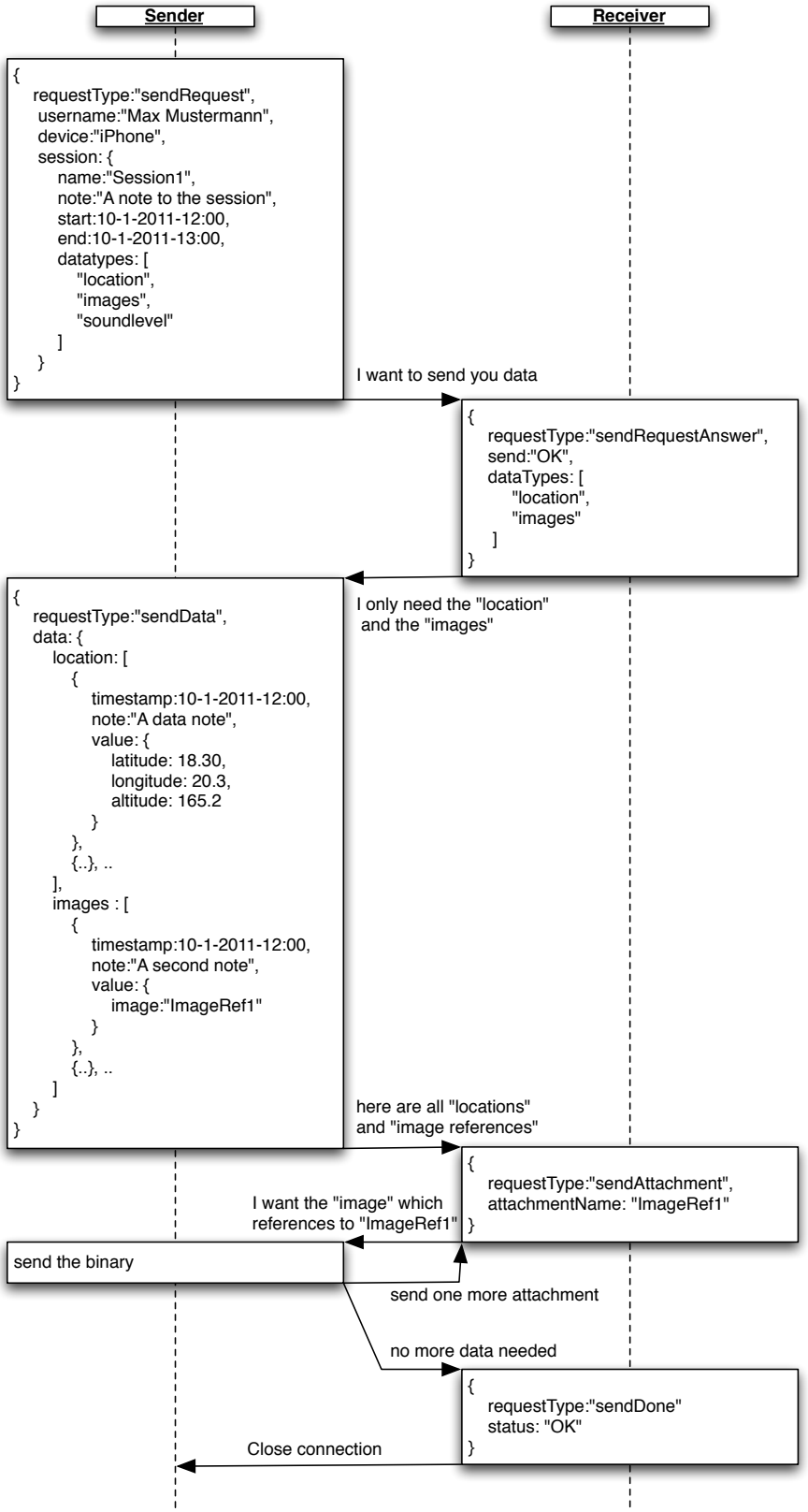

Abbildung 6.11: Netzwerk Diagramm für den Use-Case, in dem eine Instanz von iPeeper von sich aus Daten an jemanden verschicken will

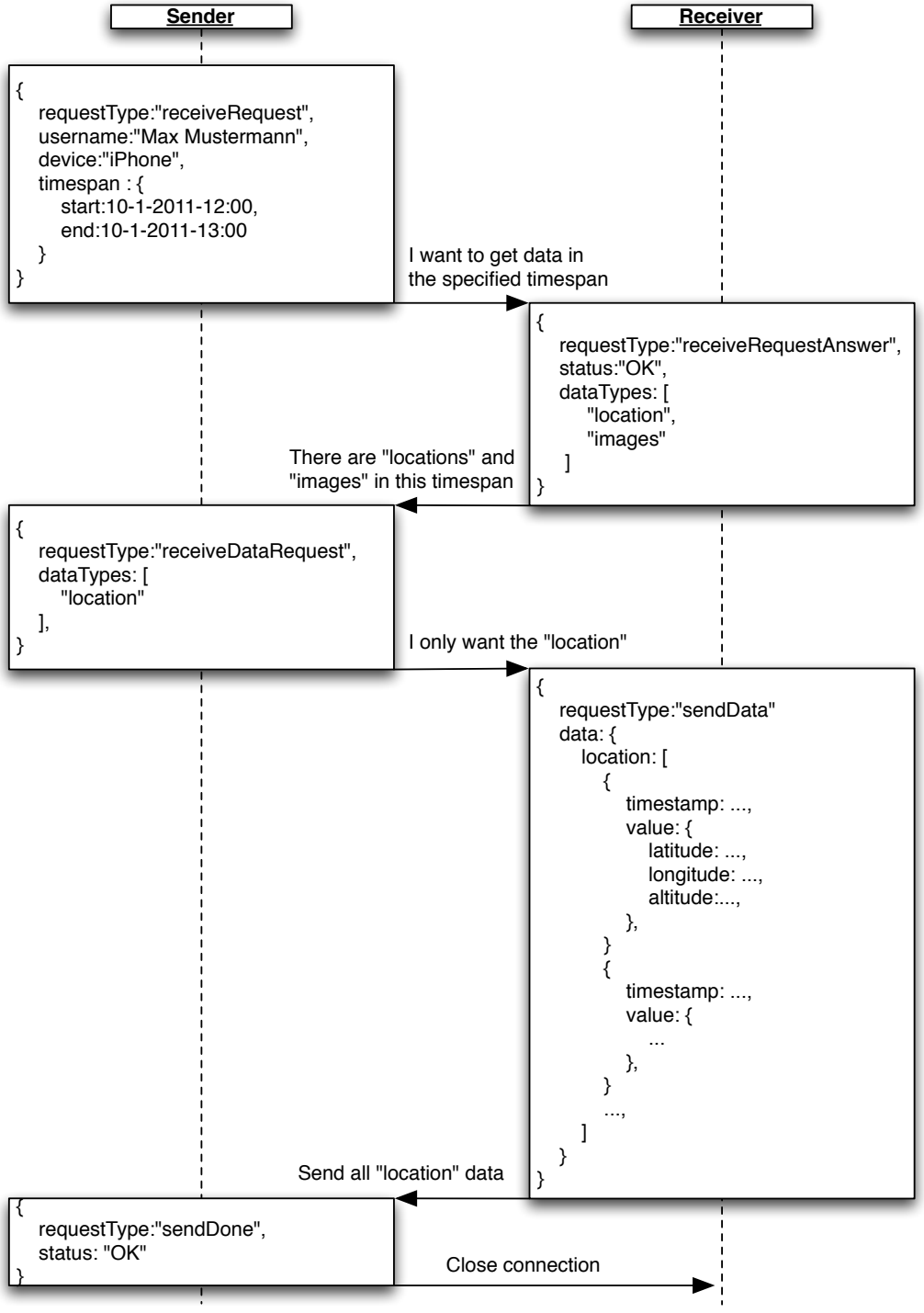

Abbildung 6.12: Netzwerk Diagramm für den Use-Case, in dem jemand Daten von einer iPeeper Instanz anfordert

Kapitel 7

Mögliche Anwendungsszenarien

"To accomplish great things we must first dream, then visualize, then plan... believe... act!" *Alfred A. Montapert*

Die iPeeper Applikation besteht aus einer Ansammlung unterschiedlichster Sensoren und Visualisierungen die je nach Anwendungsfall anders eingesetzt werden können. Aus diesem Grund macht es Sinn, die iPeeper Applikation für jeden Anwendungsfall unterschiedlich zu konfigurieren oder sogar zu verändern. Die folgenden drei Anwendungsszenarien sollen die Vielseitigkeit der Applikation zeigen, aber auch die Änderungen, die notwendig wären um iPeeper an das jeweilige Anwendungsszenario spezifisch anzupassen. Weiters werden zu jedem Szenario mögliche Weiterentwicklungen der iPeeper Applikation aufgezeigt, die speziell für dieses Szenario sinnvoll sein können.

7.1 Zivilschutz

Dieser Use Case beschreibt wie die iPeeper App, die in Kapitel 6 vorgestellt wurde, die unterschiedlichsten Einsatzorganisationen bei ihrer Arbeit unterstützen kann bzw. zeigt, wie sich diese durch Reflektion besser auf zukünftige Einsätze vorbereiten können. Im besonderen werden jedoch Einzelpersonen bzw. Personengruppen unterstützt, indem sie über ihre eigene bzw. bereits vergangene Leistung reflektieren. Durch das Reflektieren über besonders gute aber auch schlechte Leistungen, soll ein Bewusstsein für ihre Handlungen geschaffen werden. Durch diesen Vorgang sollen vergangene Einsätze nachbehandelt werden um damit zukünftige zu verbessern.

7.1.1 Szenario

Das Szenario für diesen Use Case sieht eine Großveranstaltung - wie zum Beispiel den Superbowl - vor, in der eine Einsatzorganisation für den ungestörten Ablauf der Veranstaltung verantwortlich ist. Jeder Mitarbeiter dieser Organisation bekommt von der Leitstelle einen bestimmten Bereich zugeordnet, in dem er für Ordnung sorgen soll. Diese Ordner werden dafür, zusätzlich zu ihrer Standardausrüstung, mit einem Smartphone ausgestattet auf dem die iPeeper Applikation läuft.

Im Verlauf jeder Veranstaltung kommt es bei den Ordnern zu positiven Erlebnissen, in denen sie besonders gut auf Situationen reagiert haben, aber auch zu negativen, in denen etwas nicht so gelaufen ist, wie es hätte sein sollen. All diese Ereignisse werden vollkommen automatisch mit der iPeeper App aufgezeichnet.

7.1.2 Voraussetzungen

In diesem Szenario werden Einsatzkräfte mit einem Smartphone ausgestattet, somit ist davon auszugehen, dass es sich nicht um das private Gerät einer Person handelt. Aus diesem Grund könnte es in diesem Fall nötig sein, die Daten auf irgendeine Weise zu exportieren um sich diese später auf seinem persönlichen zB. Computer ansehen zu können. Aus dem selben Grund sollten die aufgezeichneten Daten, nachdem sie anderswo gespeichert wurden, vom Gerät gelöscht werden. Deshalb könnte man in diesem Anwendungsszenario völlig auf die Möglichkeit der Sortierung durch einen Kalender verzichten und nur eine Liste an Sessions anbieten.

7.1.3 Aufzeichnung

Da jeder Einsatz unterschiedlich sein kann und aus diesem Grund auch andere Daten wichtig sein können, kann in der iPeeper App bei jeder neuen Aufzeichnung eingestellt werden, welche Sensoren aktiviert werden. Natürlich wäre es sinnvoll so viele Sensoren wie möglich zu aktivieren, jedoch können bestimmte Situationen auch das Deaktivieren von Sensoren erforderlich machen.

Ein sehr gutes Beispiel dafür wäre ein Mikrofon, welches nicht an jeder beliebigen Stelle getragen werden kann, da man ansonsten keine verwertbaren Tonaufnahmen erhält. Das Mikrofon kann somit natürlich in gewissen Situationen stören und das Einsatzteam bei der Ausübung seiner Tätigkeiten behindern. Weiters sollte bedacht werden, dass durch ein

Mikrofon sehr sensible Daten von fremden Personen aufgezeichnet werden können.

Für die Aufzeichnung in unserem Use Case werden die Sensoren

- GPS,
- Kamera,
- Umgebungslautstärke,
- Wetter,
- RSS-Feeds und
- Mood

verwendet, um ein detailgetreues Abbild des Einsatzes zu erhalten und die Situationen später bestmöglich analysieren zu können. Nach dem Starten der Sensoren muss das Smartphone so positioniert werden, dass die Kamera Bilder von der Umgebung machen kann. Ab diesem Zeitpunkt muss der Ordner nichts mehr aktiv mit dem Smartphone machen und kann sich komplett seiner Arbeit widmen. Der einzige Sensor der nur durch Benutzerinteraktion Daten sammelt ist der Mood Sensor. Damit kann der Ordner seine Gefühle bei besonders positiven oder negativen Erfahrungen zum Ausdruck bringen und direkt zu den Situation speichern.

Ein besonderer Sensor in diesem Setting ist sicherlich der RSS-Feed Sensor, da mit diesem Sensor nicht die Aktivitäten des Trägers aufgezeichnet werden, sondern Informationen, Nachrichten oder auch Befehle der Leitstelle, zu den aktuellen Aktivitäten der Person gespeichert werden können. Somit ist ein direkter Zusammenhang eines Befehles oder einer Information zu einer Aktion festzustellen und die Daten können später besser zeitlich eingeordnet werden.

7.1.4 Reflektion

Nach dem Einsatz ist es möglich sich die aufgezeichneten Daten anzusehen, zu analysieren und vor allem darüber zu reflektieren. Dies kann jeder für sich persönlich machen, oder auch in einer Gruppe, um die Meinung von anderen zu erfragen. Weiters ist es möglich seine Daten mit anderen zu teilen, um zum Beispiel neuen Mitarbeitern eine Art Handbuch zur Verfügung zu stellen und um ihnen zu zeigen, wie sie in speziellen Situationen reagieren sollten.

7.1.5 Mögliche Erweiterungen

Eine mögliche Erweiterung dieses Use Cases, wäre die Einbindung der Leitstelle als zentrale Speicher und Auswertungsstelle der aufgezeichneten Daten. Mit den gesammelten Daten, wie Aufenthaltsort, Lautstärke oder sogar Bilder könnten sie schneller Gefahrensituationen erkennen und somit auch besser darauf reagieren. Es wäre dadurch möglich die Mitarbeiter gezielt zu lenken und ihnen Anweisungen oder wertvolle Informationen über die aktuelle Situation zu geben.

7.2 IT Consulting

In diesem Use Case soll gezeigt werden wie man die iPeeper App zur Reflektion von Meetings nutzen kann. Die aufgezeichneten Daten können dabei von jeder Person einzeln oder auch von der ganzen Gruppe zur Reflektion genutzt werden.

7.2.1 Szenario

In diesem Szenario befindet sich der Benutzer in einem Meeting mit beliebig vielen Personen. Er möchte zu einem späteren Zeitpunkt über dieses Meeting reflektieren und hätte, zusätzlich zu seinen Notizen, gerne noch weitere Informationen die ihm dabei helfen. Eine sehr wichtige Information in einem Meeting, die man zu einem späteren Zeitpunkt nicht mehr wissen kann, sind die Gefühle, die man während einer bestimmten Diskussion hat. Diese Gefühle sind oft sehr schwer in Worte zu fassen und können deswegen auch nur schwer den eigenen Notizen beigefügt werden.

7.2.2 Voraussetzungen

In diesem Szenario würde es Sinn machen die Auswahl der verschiedenen Sensoren automatisch vorzunehmen. Ansonsten müsste sich der Benutzer nur zusätzlich Gedanken dazu machen, welche Sensoren er benötigt und es würde den Konfigurationsaufwand einer Session erhöhen. Weiters würde man die Live-Visualisierungen für Kamera und GPS-Sensor deaktivieren, da diese ohne zugehörige Sensoren keinen Sinn machen.

7.2.3 Aufzeichnung

Für die Aufzeichnung der Daten werden nur die beiden Sensoren

- Umgebungslautstärke und
- Mood

benötigt. Wobei beim Starten der Applikation nur der Lautstärkensensor aktiviert werden muss, da der Mood-Sensor bei jeder Aufzeichnung automatisch aktiviert wird. Der Grund dafür ist der Mood-Sensor, der seine Daten nicht automatisch aufzeichnen kann und deswegen auf die aktive Eingabe des Benutzers angewiesen ist.

Während des Meetings zeichnet nun die iPeeper Applikation laufend die Umgebungslautstärke auf, die zu einem späteren Zeitpunkt genutzt werden kann, um die Art der Diskussion einschätzen zu können. Es wird dabei darauf verzichtet ganze Gespräche aufzuzeichnen, da dies meist nicht gewünscht ist und somit zu einer Verletzung der Privatsphäre führen würde. Zusätzlich zur Umgebungslautstärke kann der Benutzer zu jeder Zeit, durch einen Klick auf die MoodMap, seine aktuellen Gefühle speichern. Da die iPeeper App diese Informationen immer mit einem Zeitstempel versieht, ist es zusätzlich sinnvoll eine Agenda, falls nicht vorhanden, mitzuführen und diese ebenfalls mit Zeitangaben zu versehen. Somit ist es später möglich die empfundenen Gefühle einzelnen Themen zuzuordnen.

7.2.4 Reflektion

Wenn der Benutzer nun über ein Meeting reflektieren möchte, sieht er sich üblicherweise seine Notizen oder auch die Agenda des Meetings durch. Zusätzlich zu diesen Informationen kann er aber auch seine Gefühle und die Lautstärke den einzelnen Gesprächsthemen zuordnen. Somit sieht er nicht nur die mitgeschriebenen Ergebnisse einer Diskussion, sondern auch seine subjektiven Gefühle zu den behandelten Themen.

7.2.5 Mögliche Erweiterungen

Eine sicherlich sehr sinnvolle Erweiterung wäre die Agenda oder bestimmte Notizen in der Applikation selbst zu speichern. Damit würde man sich das Mitschreiben direkt in seinen Notizen ersparen und der Zeitstempel würde automatisch gesetzt werden.

Der Use Case sieht vor, seine Gefühle ausschließlich für die persönliche Reflektion aufzuzeichnen. Es wäre jedoch möglich und auch sinnvoll seine Gefühle mit anderen zu teilen. Dabei sollte es sich jedoch nicht um die Gefühle einzelner Personen handeln, sondern um das kumulierte Gefühl einer ganzen Gruppe. Somit wäre es möglich während des Meetings auf die Gefühle der Gruppe einzugehen und somit ein Meeting zu verbessern. Eine Möglichkeit dafür wäre die iPeeper Applikation an das bestehende MoodMap+KnowSe System

anzubinden, das genau eine solche Funktionalität bereit stellt (siehe Kapitel 4.4.3).

7.3 Reisetagebuch

Dieser Use Case dient dazu, die Vielseitigkeit der iPeeper App zu veranschaulichen und zu zeigen, dass sie nicht nur für den Arbeitsalltag geeignet ist, sondern auch bei Freizeitaktivitäten nützlich sein kann. In diesem Use Case dient die iPeeper Applikation dazu aktuelle Informationen über eine Reise zu sammeln und diese für den späteren Gebrauch zu speichern.

7.3.1 Szenario

Der Benutzer befindet sich auf einer Reise und möchte soviel Informationen wie möglich sammeln, um sie für später zu archivieren. Der umständliche Weg wäre seinen Weg ständig auf einer Karte einzuzeichnen. Das würde bei einer Fahrt mit dem Auto noch einigermaßen funktionieren, da es sich dabei meistens um die kürzeste Strecke von einem Punkt zu einem anderen handelt und das Einzeichen leicht im Nachhinein möglich wäre. Bei einer Stadterkundung wäre das natürlich schon viel schwieriger und auch nur mit sehr hohem Aufwand möglich.

7.3.2 Voraussetzungen

In diesem Anwendungsszenario könnte die iPeeper Applikation völlig ohne Änderungen eingesetzt werden.

7.3.3 Aufzeichnung

Zur Aufzeichnung der Reisedaten muss nur die iPeeper App mit den Sensoren

- GPS,
- Kamera,
- Wetter,
- Twitter und
- RSS-Feeds

gestartet werden. Bei einem üblichen Reisetagebuch würde man hauptsächlich Bilder machen und diesen den besuchten Orten zuordnen. Mit der iPeeper App gehen wir hier noch

einen Schritt weiter und speichern zusätzlich noch das Tageswetter und verschiedenste aktuelle Informationen und Nachrichten aus unterschiedlichen zuvor festgelegten Quellen ab. Um etwas Vergleichbares auf anderem Wege zu erhalten, müsste man sich Zeitungen und Magazine kaufen und relevante Informationen daraus sammeln.

7.3.4 Reflektion

Nach einer Reise möchte man sich natürlich gerne des öfteren sein Erlebtes noch einmal ansehen. Die iPeeper App bietet hier eine hervorragende Lösung seine aufgezeichneten Daten, wie ein Video, immer wieder abzuspielen und so seine Reise immer wieder zu erleben und zu genießen.

7.3.5 Mögliche Erweiterungen

Meistens hat man auf einer Reise eine bessere Kamera, als die in das Smartphone integrierte, bei sich und möchte natürlich lieber mit dieser die Fotos aufnehmen. Da die von iPeeper aufgezeichneten Daten immer mit einem Timestamp versehen sind wäre es möglich, die Bilder einer externen Kamera mit den aufgezeichneten Sensordaten zu kombinieren.

Da die meisten Fotoverwaltungsprogramme die Möglichkeit bieten Fotos mit Koordinaten zu kombinieren, wäre eine weitere Möglichkeit der iPeeper App, die gesammelten Koordinaten zu exportieren. Somit könnte man in diesem externen Programm seine Fotos anhand des Timestamps den besuchten Orten automatisch zuordnen.

Kapitel 8

Diskussion und Ausblick

"The question is not what you look at, but what you see." *Henry David Thoreau*

8.1 Zusammenfassung

Das Ziel der vorliegenden Masterarbeit war es, ein Visualisierungs-Framework zu entwickeln, das es dem Benutzer erlaubt, Daten von mobilen Sensoren direkt auf dem selben mobilen Gerät performant und interaktiv darstellen zu können.

Die beiden zentralen Herausforderungen waren dabei, wie soeben erwähnt, zum einen das performante Darstellen der Visualisierungen am mobilen Endgerät, auf dem vor allem die Hardwareressourcen im Vergleich zu einem Desktop System stark limitiert sind. Zum anderen sollte sich das Framework schön in die gewählte Plattform integrieren. Das heißt, dass sich gewohnte User-Interaktions-Paradigmen von iOS in dem Framework widerspiegeln sollten.

Im ersten Schritt dieser Masterarbeit wurde überlegt, welche Daten ein solches Visualisierungs-Framework überhaupt darstellen können muss und welche Daten von Sensoren gängiger Smartphones zur Verfügung gestellt werden können. Weiters wurden die Probleme hinsichtlich limitierter Hardwareleistung beleuchtet, die bei der Umsetzung von interaktiven Visualisierungen auf mobilen Geräten entstehen können.

Nach einer genaueren Analyse der Grundlagen für eine interaktive Informationsvisualisierung auf mobilen Geräten, wurden Arbeiten gesucht, die ein ähnliches Ziel, nämlich

das Visualisieren von Daten, verfolgen. Anhand dieser Arbeiten wurde überlegt was davon für die Visualisierung von Sensordaten auf einem mobilen Gerät sinnvoll ist. Nach einer genaueren Untersuchung dieser ähnlichen Arbeiten, ist die Entscheidung gefallen die Visualisierungen, die im praktischen Teil dieser Masterarbeit erstellt wurden, auf sehr generische Darstellungsformen wie Linien-, Balken- bzw. Tortendiagramme, Landkarten und ähnlichem zu basieren. Der Grund dafür war zum einen der, dass es für den Benutzer einfacher ist, sich nicht immer auf neue Formen von Visualisierungen einzulernen, zum anderen sollte das Visualisierungs-Framework möglichst viele verschiedene Sensordaten entgegennehmen können und auch in Zukunft einfach erweiterbar sein. Ein anderer Aspekt war auch die limitierte Arbeitsfläche, die das Display eines Smartphones bereitstellt.

An diese ähnlichen Arbeiten angelehnt, sind daraufhin einige Visualisierungen entstanden, die es dem Benutzer ermöglichen, die Daten, die mobile Sensoren aufzeichnen, schön, übersichtlich, performant und vor allem interaktiv zu visualisieren.

Da diese Visualisierungen aber ohne “echte” Werte nicht wirklich sinnvoll sind, wurde ein Prototyp namens iPeeper entwickelt. Die Masterarbeit von [Edler, 2012] beschäftigt sich mit dem Aufzeichnen von mobilen Sensordaten, sowie dem Interpretieren dieser Werte. Aus diesem Grund machte es Sinn einen gemeinsamen Prototypen zu entwickeln. Dieser Prototyp vereint somit die Komponenten die Daten aufzeichnen, mit den Komponenten dieser vorliegenden Masterarbeit, die Daten visualisieren.

Zu guter Letzt beschreibt das letzte Kapitel dieser Masterarbeit mögliche Anwendungszwecke dieses Prototypen iPeeper.

8.2 Ergebnis

Das VisualizationKit Framework das im Zuge dieser Diplomarbeit entwickelte wurde, eignet sich sehr gut für die performante und interaktive Darstellung der derzeit bestehenden Sensoren. Es ermöglicht, dass auch große Datenmengen schnell und flüssig dargestellt werden können und vor allem Datensätze miteinander in Verbindung gesetzt werden können. Somit wurde das Hauptziel dieser Diplomarbeit meiner Meinung nach sehr gut erreicht.

Die Umsetzung der Interaktivität zwischen den einzelnen Visualisierungen in iPeeper ist vor allem am iPad sehr gut gelöst, da man hier mehrere Visualisierungen nebeneinander sieht. Am iPhone ist das aufgrund des kleineren Displays nicht ganz so komfortabel möglich.

8.3 Ausblick

Wie soeben erwähnt, wäre ein schöner Anknüpfungspunkt für iPeeper der, die Interaktivität zwischen den einzelnen Visualisierungen, vor allem am iPhone, noch weiter zu verbessern. Hier müsste man eine Lösung finden, die trotz des geringen zur Verfügung stehenden Platzes einfach zu bedienen wäre.

Eine weitere Möglichkeit wäre die konkrete Umsetzung eines der Anwendungsszenarien, die in Kapitel 7 beschrieben wurden. Mich persönlich würde speziell die Umsetzung des Anwendungsfalles interessieren, in dem iPeeper als Einsatz-Überwachung dient, da ich hier ein großes Potential für iPeeper sehen würde.

Natürlich gibt es aber auch für das Visualisierungs-Framework selbst viele mögliche Anknüpfungspunkte für weitere Arbeiten. Zum einen wäre eine mögliche Erweiterung das Erstellen weiterer Visualisierungen. Da es, wie bereits in Kapitel 6.6 beschrieben, geplant ist auch in Zukunft an iPeeper weiterzuarbeiten, entstehen sicher einige Möglichkeiten neue Darstellungsformen einzubauen, sobald weitere Sensoren neue Daten liefern. Auch eine genauere Auswertung der derzeitigen Sensordaten in iPeeper, würde neue Möglichkeiten für speziellere Visualisierungen schaffen.

Es könnten aber auch bereits bestehende Visualisierungen mehr Datenquellen akzeptieren. So wäre beispielsweise die Kartenvisualisierung ein sehr guter Startpunkt um weitere Informationen anzuzeigen. Konkret wäre das die Möglichkeit Bilder in einem Overlay darzustellen oder den Kontakt zu anderen Personen, die beispielsweise mittels Bluetooth detektiert werden könnten, darzustellen [Rudström et al., 2005].

Eine sehr schöne und willkommene Erweiterung des VisualizationKits, wäre die Möglichkeit nicht nur, wie in Kapitel 5.4 beschrieben, einen Zeitpunkt über mehrere Visualisierungen zu synchronisieren, sondern auch Zeiträume. Somit würden Visualisierungen wie das Torten- oder Balkendiagramm auch über dieses Feature verfügen und somit das ganze Framework noch interaktiver gestallten.

Literaturverzeichnis

Abowd, Gregory D., Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, und Pete Steggles [1999]. *Towards a Better Understanding of Context and Context-Awareness*. Seiten 304–307. (Zitiert auf den Seiten 1 und 26)

Andrews, Keith [2006]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Technische Universität Graz, Österreich. Online verfügbar unter `http://ftp.iicm.edu/pub/keith/thesis/`; zuletzt besucht am 24. Juli 2011. (Nicht zitiert)

Apple, Inc. [2012a]. *Bonjour Overview*. Online verfügbar unter `http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/NetServices.pdf`; zuletzt besucht am 28. März 2012. (Zitiert auf den Seiten vii, 63 und 65)

Apple, Inc. [2012b]. *Cocoa Design Patterns*. Online verfügbar unter `http://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf`; zuletzt besucht am 8. März 2012. (Zitiert auf den Seiten 39 und 54)

Apple, Inc. [2012c]. *Game Kit Programming Guide*. Online verfügbar unter `http://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/GameKit_Guide/GameKit_Guide.pdf`; zuletzt besucht am 28. März 2012. (Zitiert auf Seite 62)

Apple, Inc. [2012d]. *iOS App Programming Guide*. Online verfügbar unter `http://developer.apple.com/library/ios/DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingGuide.pdf`; zuletzt besucht am 25. März 2012. (Zitiert auf Seite 68)

Apple, Inc. [2012e]. *iOS Human Interface Guidelines*. Online verfügbar unter http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf; zuletzt besucht am 24. März 2012. (Zitiert auf den Seiten 9, 10 und 13)

Apple, Inc. [2012f]. *UIControl Class Reference*. Online verfügbar unter http://developer.apple.com/library/ios/documentation/uikit/reference/UIControl_Class/UIControl_Class.pdf; zuletzt besucht am 25. März 2012. (Zitiert auf Seite 38)

Apple, Inc. [2012g]. *UIKit Framework Reference*. Online verfügbar unter http://developer.apple.com/library/ios/documentation/uikit/reference/UIKit_Framework/UIKit_Framework.pdf; zuletzt besucht am 25. März 2012. (Zitiert auf Seite 49)

Apple, Inc. [2012h]. *UILongPressGestureRecognizer Class Reference*. Online verfügbar unter http://developer.apple.com/library/IOS/documentation/UIKit/Reference/UILongPressGestureRecognizer_Class/UILongPressGestureRecognizer_Class.pdf; zuletzt besucht am 25. März 2012. (Zitiert auf Seite 54)

Apple, Inc. [2012i]. *UIView Class Reference*. Online verfügbar unter http://developer.apple.com/library/ios/documentation/uikit/reference/uiview_class/UIView_Class.pdf; zuletzt besucht am 25. März 2012. (Zitiert auf den Seiten 38 und 45)

Borodin, Yevgen, Jalal Mahmud, und I.V. Ramakrishnan [2007]. *Context browsing with mobiles - when less is more*. *International Conference On Mobile Systems, Applications And Services*, Seite 3. (Zitiert auf den Seiten 1, 2 und 11)

Boud, David [1985]. *Reflection : Turning Experience into Learning*. Routledge, 170 Seiten. (Zitiert auf den Seiten 2, 26, 28, 55 und 67)

Eagle, Nathan und Alex Sandy Pentland [2005]. *Reality mining: sensing complex social systems*. *Personal and Ubiquitous Computing*, 10(4), Seiten 255–268. (Zitiert auf Seite 1)

Edler, Stefan [2012]. *Kontexterkennung auf mobilen Geräten*. Diplomarbeit, Technische Universität Graz, Österreich. Noch nicht veröffentlicht (Stand 1.Mai 2012). (Zitiert auf den Seiten 3, 4, 56, 60, 68 und 80)

Google, Inc. [2012]. *User Interface Guidelines | Android Developers*. Online verfügbar unter `http://developer.android.com/guide/practices/ui_guidelines/index.html`; zuletzt besucht am 24. März 2012. (Zitiert auf den Seiten 10 und 13)

Hamming, Richard Wesley [1987]. *Numerical methods for scientists and engineers*. Zweite Auflage. Dover Publications. (Zitiert auf Seite 1)

Hodges, Steve, Emma Berry, und Ken Wood [2011]. *SenseCam: A wearable camera that stimulates and rehabilitates autobiographical memory*. *Memory*, 19(7), Seiten 685–696. (Zitiert auf Seite 6)

Itten, Johannes [2003]. *Kunst der Farbe. Subjektives Erleben und objektives Erkennen als Wege zur Kunst.* Urania-Verl., 95 Seiten. (Zitiert auf den Seiten 28 und 48)

Keim, Daniel A. und Daniela Oelke [2007]. *Literature Fingerprinting: A New Method for Visual Literary Analysis*. In *Proceedings of the 2007 IEEE Symposium on Visual Analytics Science and Technology*, Seiten 115–122. VAST '07, IEEE Computer Society, Washington, DC, USA. (Zitiert auf den Seiten vi, 32 und 33)

Kerren, Andreas, John T. Stasko, Jean-Daniel Fekete, und Chris North [2008]. *Information Visualization*, *Lecture Notes in Computer Science*, Band 4950. Springer Berlin Heidelberg, Berlin, Heidelberg, 154–175–175 Seiten. (Zitiert auf den Seiten v, 11 und 12)

Lane, Nicholas D, Mashfiqui Mohammod, Mu Lin, Xiaochao Yang, Hong Lu, Shahid Ali, Afsaneh Doryab, Ethan Berke, Tanzeem Choudhury, und Andrew T Campbell [2011]. *BeWell : A Smartphone Application to Monitor , Model and Promote Wellbeing*. *Health Policy*. (Zitiert auf den Seiten v, 22 und 24)

Lengler, Ralph und Martin J. Eppler [2012]. *A periodic table of visualizations methods*. Online verfügbar unter `http://www.visual-literacy.org/periodic_table/periodic_table.html`; zuletzt besucht am 27. März 2012. (Zitiert auf den Seiten v und 14)

Lu, Hong, Wei Pan, Nicholas D. Lane, Tanzeem Choudhury, und Andrew T. Campbell [2009]. *SoundSense: scalable sound sensing for people-centric applications on mobile phones*. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, Seiten 165–178. MobiSys '09, ACM, New York, NY, USA. (Zitiert auf Seite 5)

Maniar, Nipan, Emily Bennett, Steve Hand, und George Allan [2008]. *The Effect of Mobile Phone Screen Size on Video Based Learning*. *Journal of Software*, 3(4), Seiten 51–61. (Zitiert auf Seite 10)

Neuburg, M. [2011]. *Programming iOS 4: Fundamentals of iPhone, iPad, and iPod touch Development*. Definitive Guide Series, O'Reilly Media. (Zitiert auf Seite 38)

Rath, Andreas [2010]. *User Interaction Context - Studying and Enhancing Automatic User Task Detection on the ComputerDesktop via an Ontology-based User Interaction Context Model*. Diplomarbeit, Technische Universität Graz, Österreich. Online verfügbar unter `http://know-center.tugraz.at/wp-content/uploads/2010/12/Dissertation_Andreas_Rath.pdf`; zuletzt besucht am 10. April 2012. (Zitiert auf den Seiten 1, 26, 29 und 62)

Rath, Andreas, Nicolas Weber, Mark Kröll, Olivia Dietzel, Micheal Granitzer, und Stefanie N. Lindstaedt [2008]. *Context-Aware Knowledge Services*. *Event (London)*. (Zitiert auf Seite 1)

Rudström, Å sa, Kristina Höök, und Martin Svensson [2005]. *Social positioning: Designing the Seams between Social , Physical and Digital Space Social Positioning of Information in Seamful Designs*. *Science*. (Zitiert auf den Seiten 7 und 81)

Russell, James A. [1980]. *A circumplex model of affect. Journal of Personality and Social Psychology*, 39(6), Seiten 1161–1178. (Zitiert auf den Seiten vi, 28, 47 und 48)

Seifert, Josef W. [2003]. *Visualisieren, Präsentieren, Moderieren*, Band 23. GABAL-Verlag GmbH, 173 Seiten. (Zitiert auf Seite 4)

Shilton, Katie [2009]. *Four billion little brothers? Communications of the ACM*, 52(11), Seite 48. (Zitiert auf Seite 51)

Shneiderman, Ben [2008]. *Extreme visualization*. ACM Press, New York, New York, USA, 3–12 Seiten. (Zitiert auf den Seiten vi, 2, 11, 30, 31 und 32)

Ståhl, Anna und Kristin Höök [2006]. *Experiencing the Affective Diary*. Personal and Ubiquitous Computing. (Zitiert auf den Seiten v, 26 und 27)

Ståhl, Anna, Petra Sundström, und Kristina Höök [2005]. *A foundation for emotional expressivity*. In *Proceedings of the 2005 conference on Designing for User eXperience*.

DUX '05, AIGA: American Institute of Graphic Arts, New York, NY, USA. (Zitiert auf den Seiten 28 und 48)

Sundström, Petra, Anna Ståhl, und Kristina Höök [2007]. *In situ informants exploring an emotional mobile messaging system in their everyday practice*. International Journal of Human-Computer Studies. (Zitiert auf Seite 47)

Wilkinson, Leland [2005]. *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. (Zitiert auf den Seiten v und 15)

Wood, Ken, Rowanne Fleck, und Lyndsay Williams [2004]. *Playing with sensecam*. *Proc Playing with Sensors W3 at UbiComp 2004*, Seiten 2–3. (Zitiert auf Seite 6)

Zeqian, Shen und Ma Kwan-Liu [2008]. *MobiVis: A Visualization System for Exploring Mobile Data*. *2008 IEEE Pacific Visualization Symposium*, Seiten 175–182. (Zitiert auf den Seiten v, 26 und 27)