Aufzeichnung von mobilen Sensordaten

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Stefan Edler

Institut für Wissensmanagement (KMI),

Technische Universität Graz

A-8010 Graz

12. September 2012

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

Betreuerin: Dr. Viktoria Pammer-Schindler

Mobile Phone Sensing

Master's Thesis

at

Graz University of Technology

submitted by

Stefan Edler

Knowledge Management Institute (KMI),
Graz University of Technology
A-8010 Graz, Austria

September 12, 2012

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

Advisor: Dr. Viktoria Pammer-Schindler

Abstract

Mobile phone sensing has started as an area of research and development when mobile phones became smartphones due to the different technological improvements. Research in this area, referring to the amount of publications, is growing very fast and it is not restricted to a large research group. Mobile phone sensing is attractive for research because no special equipment is needed and due to the different available app stores the distribution of the apps gets easier.

This master thesis deals with the recording and storing of mobile sensor data. It evaluates the possibilities of the current available internal sensors of an iOS device like an iPhone or iPad. This evaluation results in a framework for recording and storing mobile sensor data on iOS devices. The framework takes care of the biggest challenge in developing for mobile devices. It deals with the existing mobile phone restrictions like limited memory and performance.

Furthermore a prototype named iPeeper was implemented to demonstrate the possibilities of the framework. The prototype records, stores and visualizes environmental sound, images from the internal camera, activities, moods, GPS coordinates, weather data based on these coordinates, RSS-Feeds and Twitter activities. It is also possible to share these recorded sensor data with other devices.

Keywords: mobile, context, sensors, data, record, store

Kurzfassung

Das „Mobile Phone Sensing“ enstand als sich die Mobiltelefone weiterentwickelt haben und zu den heutigen Smartphones wurden. Möglich wurde dieser neue Forschungsbereich durch zahlreiche technologische Fortschritte in der Entwicklung von mobilen Geräten. Forschungen in diesem Gebiet werden, wenn man sich die Anzahl der Publikation der letzten Jahre ansieht, immer häufiger und sind nicht zwangsläufig auf große Forschungsgruppen beschränkt. Da keine spezielle Ausrüstung benötigt wird und das Verteilen, aufgrund der verschiedenen App Stores, sehr einfach ist, können auch kleinere Einrichtungen in diesem Bereich forschen.

Diese Masterarbeit beschäftigt sich mit der Aufzeichnung und Speicherung von mobilen Sensordaten. Dabei wurden die Möglichkeiten der internen Sensoren von iOS Geräten wie iPhone oder iPad ausgewertet. Daraus entstand ein Framework zur Aufzeichnung und Speicherung von mobilen Sensordaten auf iOS Geräten. Das Framework berücksichtigt dabei die größte Herausforderung in der Entwicklung für mobile Geräte. Es geht dabei auf die bestehenden Einschränkungen von iOS Geräte wie limitierter Speicher und Leistung ein.

Um die Möglichkeiten des Frameworks zu zeigen wurde, zusammen mit einem Framework zur Visualisierung von mobilen Sensordaten, der Prototyp iPeeper entwickelt. Dieser Prototyp zeigt, unter Einhaltung iOS typischer Paradigmen, das Aufzeichnen, Speichern und Visualisieren von unterschiedlichen Sensordaten. Dabei unterstützt er Umgebungsgeräusche, Bilder der integrierten Kamera, Aktivitäten, Gefühle, GPS Koordinaten, Wetter Daten basierend auf diesen Koordinaten, RSS-Feeds und Twitter Aktivitäten. Weiters ist es möglich diese Sensordaten, nach dem Aufzeichnen und Speichern, mit anderen Geräten auszutauschen.

Schlagworte: Mobil, Context, Sensoren, Daten, Aufzeichnung, Speicherung

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagungen

An dieser Stelle möchte ich mich bei all jenen bedanken, die mir beim Erstellen dieser Arbeit durch ihre Unterstützung zur Seite gestanden sind.

Allen voran bei meiner Frau Sandra und bei meinen Kindern, Maya und Mona, da sie viel Geduld mit mir hatten und mir die nötige Zeit gegeben haben, diese Arbeit zu schreiben. Weiters möchte ich mich bei meiner Familie und bei meinen Freunden, im besonderen bei meinem besten Freund und Kollegen Georg Bachmann, bedanken. Sie alle haben mich stets unterstützt und mir die nötige Motivation gegeben weiter zu machen.

Zu guter Letzt danke ich Frau Dr. Viktoria Pammer-Schindler, die mir als Betreuerin immer hilfreich zur Seite gestanden ist und Frau Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt für die Möglichkeit meine Masterarbeit am Know-Center schreiben zu können.

Stefan Edler

Graz, Austria, September 2012

Kapitel 1

Einleitung

1.1 Motivation und Herausforderungen

Der Bereich „Mobile Phone Sensing“ oder auch Kontexterkennung auf mobilen Geräten ist ein sehr junger Forschungszweig und erfreut sicher immer größerer Beliebtheit. In den letzten Jahren wurde diese Forschungsbereich, ausgelöst durch die zahlreichen technologischen Fortschritte, immer interessanter und auch für kleinere Forschungsgruppen leichter zugänglich. Man benötigt keine spezielle Ausrüstung mehr, sondern kann mit alltäglichen Smartphones hervorragende Ergebnisse erzielen.

1.2 Zielbeschreibung

In dieser Masterarbeit soll untersucht werden inwieweit aktuelle Consumer-Produkte genutzt werde können um Sensordaten aufzuzeichnen. Weiters werden die, trotz der enormen Fortschritte der letzten Jahre, weiterhin bestehenden Einschränkungen mobiler Geräte untersucht. Ausgehend von diesen Untersuchungen soll ein Framework zur Aufnahme von Sensordaten entwickelt werden. Dieses Framework soll mit den am aktuellen iPhone gängigen Sensoren ausgestattet sein, jedoch auch die Möglichkeit bieten dieses um zukünftige Sensoren zu erweitern.

Die Daten der Sensoren sollen jedoch nicht nur angezeigt sondern zusätzlich für andere Zwecke gespeichert werden. Diese Speicherung und natürlich auch das spätere Abrufen dieser Daten soll, unter Berücksichtigung der gegebenen Einschränkungen, effizient möglich sein. Um die Daten, zum Beispiel später für eine Visualisierung, die jedoch nicht Teil

dieser Masterarbeit ist, weiter nutzen zu können müssen offene Schnittstellen für andere Programmteile zur Verfügung gestellt werden.

Um die Möglichkeiten und Einsatzzwecke dieses Frameworks zu zeigen soll das Framework noch in einem Prototyp Verwendung finden. Dieser Prototyp soll zusätzlich zu den Funktionalität des Frameworks noch die Möglichkeit bieten die Daten entsprechend zu visualisieren. Weiters sollen die aufgezeichneten Daten mit anderen geteilt werden können. Diese Funktion setzt jedoch eine Komponente voraus, die die Netzwerkverbindung sowie den Datentransfer übernimmt.

1.3 Gliederung

Kapitel 1: Einleitung

Im Einleitungskapitel wird eine kurze Übersicht über die Masterarbeit gegeben und auch auf das Umfeld dieser eingegangen. Es wird weiters der Zweck und auch die Probleme, die diese Masterarbeit versucht zu lösen, beschrieben.

Kapitel 2: Mobile Phone Sensing

Dieses Kapitel beschäftigt sich mit den Grundlagen des „Mobile Phone Sensing“ und damit auch mit den Grundlagen dieser Masterarbeit. Es werden die aktuellen Entwicklungen im mobilen Bereich aufgezeigt und die Technologien beschrieben die diese Entwicklung erst möglich machten. Danach wird noch eine generische Architektur vorgestellt auf die zukünftige Entwicklungen aufbauen könnten. Im speziellen wird hierbei aber die Bereiche „Aufzeichnen“, „Lernen, Interpretieren“ und „Datenverwertung“ eingegangen.

Kapitel 3: Implementation

Um einen Einblick in die Schwierigkeiten der mobilen Entwicklung zu geben wird in diesem Kapitel zuerst auf die Einschränkungen der mobilen Entwicklung eingegangen. Danach werden die beiden Frameworks die im Zuge dieser Masterarbeit entstanden sind genauer beschrieben. Dadurch wird ein Einblick in die Architektur gegeben um sich dann auch direkt im Code auszukennen und die Vorgänge besser verstehen zu können.

Kapitel 4: Ähnliche Arbeiten

In diesem Kapitel werden Applikationen und Projekte beschrieben die Ähnlichkeiten zu dieser Masterarbeit haben und sich somit ebenfalls mit „Mobile Phone Sensing“ beschäftigen.

Kapitel 5: Prototyp - iPeeper

Im Zuge dieser Masterarbeit ist in Zusammenarbeit mit Georg Bachmann ein Prototyp entstanden, welcher sich mit dem Aufzeichnen, Speichern und Visualisieren von mobilen Sensor Daten beschäftigt. Grund für diese Zusammenarbeit ist Georg Bachmann's Masterarbeit die sich mit der Visualisierung von mobilen Sensordaten beschäftigt und somit eine perfekte Ergänzung zu dieser Masterarbeit bietet. Diese Kapitel ist in Zusammenarbeit mit Georg Bachmann enstanden und kommt somit in beiden Masterarbeiten beinahe wortgleich vor. Das Kapitel gibt einen Überblick über die entstandene Applikation und zeigt anhand dieser Applikation die Möglichkeiten der entstandenen Frameworks. [Bachmann, 2012]

Kapitel 6: Mögliche Anwendungsszenarien

Dieses Kapitel beschreibt drei völlig unterschiedliche Anwendungsszenarien in welche der Prototyp und ihre Frameworks noch eingesetzt werden können. Dadurch soll die Vielseitigkeit der entstanden Frameworks und damit des Prototypen aufgezeigt werden. Dafür werden die Szenarien genauestens beschrieben und auch darauf eingegangen welchen Nutzen der Benutzer daraus zieht. Zusätzlich werden noch Erweiterungen besprochen die nur durch einen Weiterentwicklung der Frameworks möglich wäre. Da sich dieses Kapitel ebenfalls auf den, gemeinsam mit Georg Bachmann entwickelten, Prototypen bezieht kommt es ebenfalls in beiden Masterarbeiten wortgleich vor.

Kapitel 7: Zusammenfassung

Zum Ende dieser Masterarbeit werden die Ergebnisse nochmals kurz zusammengefasst. Zusätzlich werden noch zukünftige Weiterentwicklungen besprochen die sicherlich einen Mehrwert zum aktuellen Stand bringen würden.

Kapitel 2

Mobile Phone Sensing

Wenn man sich die aktuellen Verkaufszahlen, in Abbildung 2.1, von Smartphones ansieht wird man feststellen, dass im Jahr 2011 mit weltweit 488 Millionen Smartphones, erstmals mehr Smartphones verkauft wurden als PCs, Netbooks und Tablets zusammen, die insgesamt nur auf 415 Millionen Geräte kommen. Dieser rasante Anstieg deutet darauf hin, dass Smartphones in unserem Leben immer mehr an Bedeutung gewinnen und zu unseren zentralen Computern und Kommunikationsgeräten werden [Canalys, 2012]. Glaubt man [Lane et al., 2010] werden Mobiltelefone, die mit Sensoren ausgestattet sind, sogar eine Revolution in vielen Bereichen wie Wirtschaft, Gesundheit, soziale Netzwerke und Logistik auslösen.

Worldwide smart phone and client PC shipments

Shipments and growth rates by category, Q4 2011 and full year 2011

Category	Q4 2011 shipments (millions)	Growth Q4'11/Q4'10	Full year 2011 shipments (millions)	Growth 2011/2010
Smart phones	158.5	56.6%	487.7	62.7%
Total client PCs	120.2	16.3%	414.6	14.8%
- Pads	26.5	186.2%	63.2	274.2%
- Netbooks	6.7	-32.4%	29.4	-25.3%
- Notebooks	57.9	7.3%	209.6	7.5%
- Desktops	29.1	-3.6%	112.4	2.3%

Source: Canalys estimates © Canalys 2012

Abbildung 2.1: Verkaufszahlen für Smartphones und PCs [Canalys, 2012]

Aufgrund dieser Entwicklung erfreut sich das Aufzeichnen und Sammeln von Daten mithilfe von Smartphones immer größerer Beliebtheit. In der Forschung bezeichnet man mit

„Mobile Phone Sensing“, das auch „Urban Sensing“ oder „Participatory Sensing“ genannt wird, das Sammeln von Daten für persönliche oder soziale Projekte mithilfe von Smartphones [Shilton, 2009] [Cuff et al., 2008]. Es soll jedem ermöglichen und es sogar noch fördern, dass er zuvor nicht erkennbare bzw. einsehbare Daten sammeln und erforschen kann. Dabei soll der Benutzer weder gezwungen werden seine Daten aufzuzeichnen noch soll es heimlich geschehen. Vielmehr soll er aktiv in den Prozess des Aufnehmens eingebunden werden [Eisenman et al., 2006] [Eagle, 2008].

Forschungen die in die selbe Richtung wie „Mobile Phone Sensing“ gingen waren früher aufwändiger und nur für eine kleine Personengruppe ausgerichtet [Miluzzo et al., 2010]. Ein dem heutigen Smartphone ähnliches Gerät, bezogen auf seine Sensoren, ist zum Beispiel die „Mobile Sensing Platform“ [Choudhury et al., 2008]. Der größte Nachteil eines solchen Gerätes liegt jedoch in seiner physischen Existenz. Niemand möchte zusätzlich zu seinem Mobiltelefon, das sowieso jeder bei sich trägt, ein zusätzliches Gerät mit sich führen. Ein weiterer Nachteil solcher separater Geräte ist natürlich auch die geringe Verbreitung und die damit verbundene Anzahl an Benutzern. Falls diese Sensorgeräte weiter verbreitet wären, müsste man sich zusätzlich noch um die Verteilung von Applikationen kümmern, die die gewonnen Informationen entgegen nehmen, aufbereiten, visualisieren oder auch versenden. Aus diesen Gründen werden solche Forschungsprototypen meist nur in sehr kleinen Personengruppen und über einen festgelegten Zeitraum genutzt [Miluzzo et al., 2010].

Daraus ergibt sich die logische Schlussfolgerung, Geräte wie die „Mobile Sensing Platform“, soweit es möglich ist, durch moderne Smartphones zu ersetzen. Ein Mobiltelefon trägt heutzutage beinahe jeder, zu jeder Zeit, sowieso bei sich und die alten Mobiltelefone werden immer mehr von, mit Sensoren angereicherten, Smartphones verdrängt [Canalys, 2012]. Damit entfällt die Notwendigkeit ein weiteres Gerät mit sich herumzutragen und zusätzlich steigt die Anzahl der potenziellen Benutzer immer weiter.

2.1 Technologien

Im Bereich der Mobiltelefone wurden unzählige Fortschritte erreicht, um aus den vergangenen Mobiltelefonen die Smartphones die wir heute kennen zu entwickeln. Ohne diese Entwicklungen wäre das „Mobile Phone Sensing“ undenkbar. Im folgenden werden nun die, laut [Lane et al., 2010], fünf wichtigsten technologischen Entwicklungen, die unabhängig vom Hersteller in allen Smartphones vorkommen, näher erläutert.

2.1.1 Applikationen

Aktuelle Smartphones leben nicht nur von schneller Hardware, tollem Design oder einem brillanten Display. Vor allem leben sie von den zahlreichen Apps die bereits zum größten Teil von Drittherstellern entwickelt wurden. Vor der Zeit der Smartphones fehlte die Unterstützung von nachträglich installierten Applikationen bzw. war dies nur sehr eingeschränkt möglich. Durch die aktive Unterstützung der Hersteller mit Entwicklungsumgebungen, SDKs, Dokumentationen und Codebeispielen wurden den Software- und auch Hardwareentwicklern sehr viele Einstiegshürden genommen. Der leichte Einstieg und natürlich auch der enorme Erfolg der Smartphones treibt die Entwicklung weiterer Applikation, wie in Abbildung 2.2 zu sehen ist, immer weiter voran.

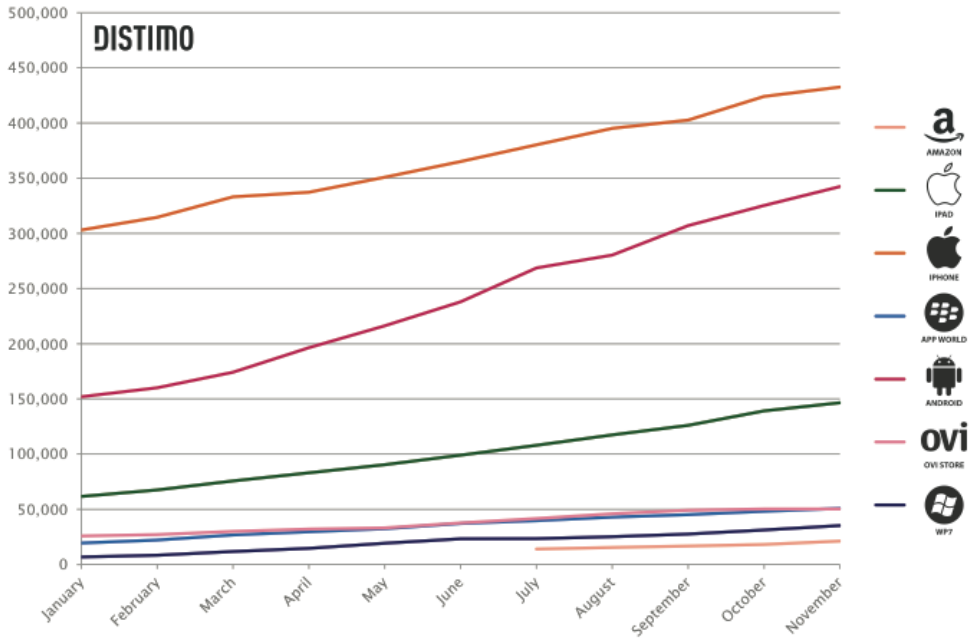


Abbildung 2.2: Verfügbare Applikation pro Monat im Jahr 2011 in den USA [Koekkoek, 2011]

2.1.2 APIs

Jedes aktuelle Smartphone Betriebssystem bietet zahlreiche APIs die von den Entwicklern genutzt werden können. Somit ist es leichter möglich für einen Software- oder Hardwareentwickler diese APIs anzusprechen und für seine eigenen Zwecke zu nutzen. Wenn man zum Beispiel den GPS-Sensor eines iOS Gerätes betrachtet, bietet dieser eine API mit der man sich bei einem CLLocationManager registrieren kann. Dieser CLLocationManger kann

nach eigenen Bedürfnissen konfiguriert werden und informiert die registrierte Applikation bei einer Änderung der GPS-Koordinaten automatisch. Durch solche, vom Betriebssystem Hersteller bereitgestellten APIs, werden den Entwicklern viele Stunden die für eine Eigenentwicklung aufgewendet werden müssten abgenommen.

2.1.3 AppStores

Wie die Tabelle 2.1 zeigt, bietet bereits jeder Hersteller einen eigenen AppStore an. Die Möglichkeit seine Applikationen über einen AppStore zu vertreiben nimmt dem Entwickler einen großen Administrationsaufwand ab. Er muss sich nicht mehr um die Bereitstellung seiner Applikation auf der eigenen Webseite kümmern und was den meisten Entwicklern sicherlich am wichtigsten ist, die Zahlungsabwicklung wird komplett von den AppStores übernommen. Des weiteren werden die Applikationen in einem AppStore eher gefunden, als auf einer Webseite irgendwo im Internet.

Logo	Hersteller	Start	Plattformen	Gewinn	Gebühren
	Apple	Juli, 2008	iOS	70 %	USD 99
	Google	Oktober, 2008	Android OS	70 %	USD 25
	RIM	April, 2009	BlackBerry	70 %	USD 0
	Nokia	Mai, 2009	Flash Lite, Java, Maemo, Symbion, WRT Widgets	70 %	EUR 1
	HP (zuvor Palm)	Juni, 2009	HP Web OS	70 %	USD 0
	Microsoft	Oktober, 2009	Windows Phone OS	70 %	USD 19,99 (5x kostenlos)

Tabelle 2.1: Übersicht der aktuellen App Stores

2.1.4 Sensoren

Durch die Verfügbarkeit von billigen Sensoren, wurden diese anfangs in die Smartphones integriert, um die User Experience zu verbessern. So wird zum Beispiel der Beschleunigungssensor dazu verwendet die Orientierung eines Smartphones zu bestimmen und somit den Bildschirm in Portrait oder Landscape darzustellen. Ein weiteres Beispiel ist der Näherungssensor der automatisch den Bildschirm deaktiviert, und somit keine ungewollten Aktionen ausführt, wenn man das Smartphone zum Telefonieren ans Ohr hält.

Weiters wird die Funktionalität, die sich hinter einer API-Methode versteckt, von den Herstellern weiterentwickelt und an aktuelle Versionen angepasst. Da sich nur die Funktionalität ändert, nicht aber die API-Methode, entsteht für den Entwickler meist kein oder nur ein geringer Aufwand wenn ein neues Betriebssystemupdate veröffentlicht wird.

2.1.5 Cloud Services

Cloud Services wurden in den letzten Jahren zu einem immer größer werdenden Hype. Immer mehr Privatpersonen, aber auch Unternehmen, vertrauen auf die Speicherung ihrer Daten in der Cloud. Laut [Avanade, 2011] haben Cloud Services, im Jahr 2011, nun ihren ersten Milestone als voll entwickelte Technologie erreicht. Avanade definiert dabei folgende Kennzeichen die für die Erreichung des Milestones von Bedeutung sind.

- Erhöhte Investitionen in Ressourcen um Cloud Services zu unterstützen und zu verwalten. Weiters wurde in die Sicherheit, die bei Cloud Services am meisten kritisiert wird, investiert.
- Cloud Services erfreuen sich einer immer größer werdenden Akzeptanz. Es geht mittlerweile so weit, dass Applikationen, die Cloud Services unterstützen sogar bevorzugt werden.
- Unternehmen entdecken die Cloud immer mehr als Einnahmequelle und versuchen damit Produkte zu realisieren bzw. ihre bestehenden Produkte damit zu erweitern.

Basierend auf einer aktuellen Studie, durchgeführt an 573 C-Level-Führungskräften, Leitern von Unternehmenseinheiten und IT-Entscheidungsträgern, wurde die Abbildung 2.3 erstellt, die die aktuelle Entwicklung im Bereich Cloud Services darstellt.

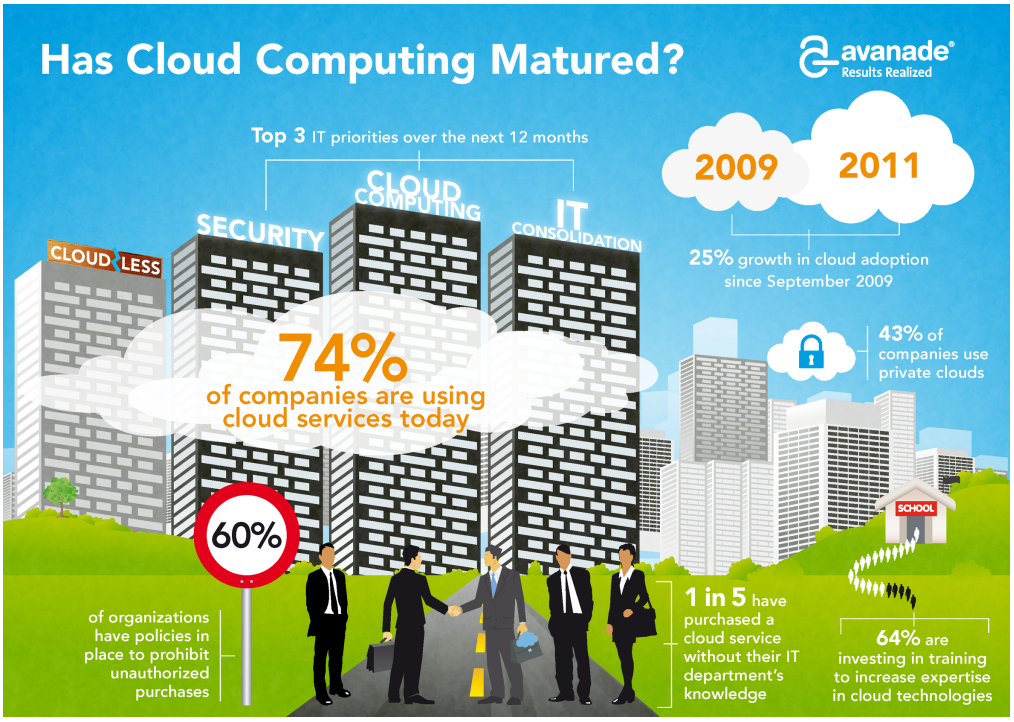


Abbildung 2.3: Entwicklung von Cloud Services [Avanade, 2011]

2.2 Generische Architektur

Da es sich bei „Mobile Phone Sensing“ um ein relativ neues Forschungsgebiet handelt, gibt es laut [Lane et al., 2010] noch keine etablierte Architektur die bevorzugt eingesetzt werden sollte. Man ist sich zum Beispiel noch nicht einige welche Daten direkt auf dem Smartphone verarbeitet werden sollten und welche Daten besser von einem Cloud Service. Die Überlegungen in diesem Bereich gehen von übermäßigen Batterieverbrauch bei direkter Verarbeitung auf dem Gerät bis zu Sicherheitsbedenken bei Cloud Lösungen.

Die im folgenden vorgestellte Architektur stammt von [Lane et al., 2010] und ist nur ein grober Versuch einer möglichen Architektur. Sie soll lediglich als Ausgangspunkt für zukünftige Entwicklungen dienen. Wie in Abbildung 2.4 zu sehen ist beschreibt die Architektur einen Kreislauf von Daten. Smartphones werden hier in erster Linie zur Gewinnung von Sensordaten und zur Speicherung oder auch Verteilung dieser Daten benötigt. Weiters kann es möglich sein, dass anhand von Machine Learning Algorithmen oder Data Mining, eine Abstraktion der Sensordaten direkt auf dem Gerät durchgeführt wird. Die Sensordaten oder auch die daraus gewonnenen Informationen werden dann an einen oder mehrere Cloud Services verteilt. Dieser könnte die Daten weiter verarbeiten, auswerten, speichern oder auch wieder Smartphones oder Cloud Services zur Verfügung stellen.

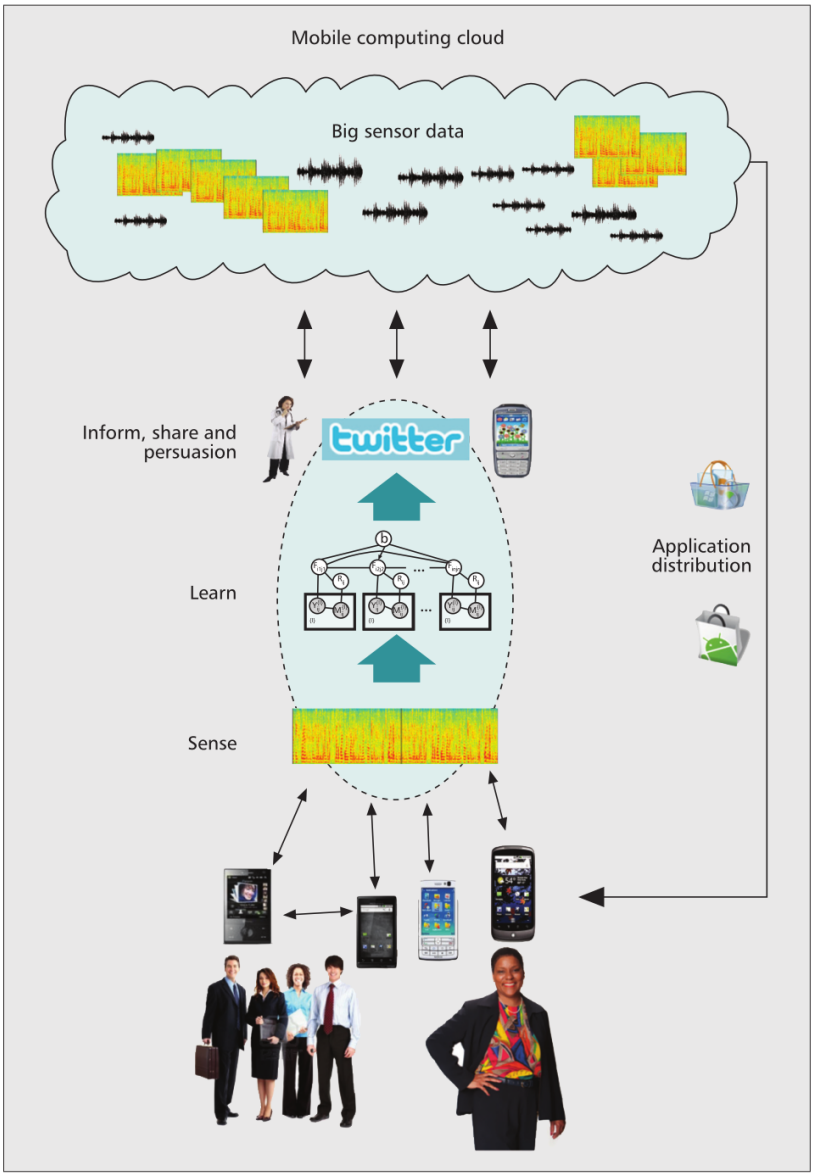


Abbildung 2.4: Mobile Phone Sensing Architektur [Lane et al., 2010]

Im folgenden Teil wird nun nochmals genauer auf die drei Hauptkomponenten

- Aufzeichnung,
- Dateninterpretierung und
- Datenverwertung (Informieren, Teilen und Speichern)

eingegangen. Dabei wird vor allem auf die Probleme der einzelnen Komponenten aber auch auf mögliche Lösungen eingegangen.

2.2.1 Aufzeichnung

Bis vor kurzem gab es noch sehr wenige Mobiltelefone auf dem Markt, die über integrierte Sensoren verfügten. Die Ersten mit integrierten Sensoren und einer Möglichkeit Applikationen zu installieren waren mit dem Nokia Betriebssystem Symbian ausgestattet. Dieses Betriebssystem stellte Entwickler jedoch einige Hürden in den Weg, da es an verlässlichen Interfaces zu den Low-Level Sensoren mangelte. Außerdem war die Leistung dieser Geräte noch nicht ausreichend um Anspruchsvolle Daten- bzw. Signalverarbeitung am Gerät durchzuführen. Erste Geräte mit eingebauten Sensoren, wie zum Beispiel dem Nokia N80 mit eingebautem Beschleunigungssensor, fehlte es an offenen Schnittstellen, um mit ihnen Arbeiten zu können. Diese Umstände haben sich jedoch in den letzten Jahren drastisch verbessert und ermöglichen nun ein Aufzeichnen unterschiedlichster Sensoren über wohl definierte Schnittstellen.

Trotz dieser positiven Entwicklungen gibt es natürlich nach wie vor einige Einschränkungen bei mobilen Geräten. Das größte Problem für „Continous Sensing“ ist jedoch die fehlende Unterstützung von echtem Multitasking. Diese Einschränkung betrifft nicht mehr jedes Betriebssystem, doch ohne Multitasking wäre eine Aufzeichnung nur möglich wenn die Applikation aktiv im Vordergrund läuft.

Der Grund, warum sich Hersteller aktiv gegen Multitasking aussprechen, liegt darin, dass es bei mehreren aktiven Applikationen zu einer höheren Auslastung kommt. Die erhöhte Auslastung schlägt sich natürlich unmittelbar, in der für Smartphones sehr wichtigen, Akkulaufzeit nieder. Laut [Miluzzo et al., 2008] kann die Akkulaufzeit im Standby dabei von 20 Stunden und mehr auf nur mehr 6 Stunden verkürzt werden. Eine Möglichkeit diesem Problem entgegen zu wirken wäre eine bessere Kontrolle der internen Sensoren. Zurzeit laufen die Sensoren in einer Art BlackBox und können nur über offene Schnittstellen angesprochen werden. Hätte man besseren Zugriff auf die Sensoren, wäre es zum Beispiel möglich diese Akku schonender einzusetzen. Aktuelle Ansätze um Strom zu sparen gehen jedoch in eine andere Richtung. Möglichkeiten wären zum Beispiel das Auslagern von sehr rechenintensiven Prozessen in die Cloud [Cuervo et al., 2012] oder auch die Reduzierung der Genauigkeit von Sensoren.

Eine völlig andere Richtung schlägt das Little Rock Projekt von Microsoft Research ein. In diesem Projekt wird ein eigenständiger, sehr energiesparender Prozessor zur Unterstützung von Langzeitaufzeichnungen entwickelt. Dieser soll, wie in Abbildung 2.5 zu sehen ist, parallel zum Hauptprozessor des Mobiltelefons laufen und ausschließlich für das Samp-

ling und die Verarbeitung zuständig sein. Wenn der Little Rock Prozessor eine Veränderung der Sensordaten feststellt, die einer weiteren Bearbeitung unterzogen werden müssen, wird der Smartphone Prozessor davon informiert. Damit ist es möglich den Smartphone Prozessor in eine energiesparenden Modus zu versetzten und nur zu wecken wenn es wirklich nötig ist. [Priyantha et al., 2010]

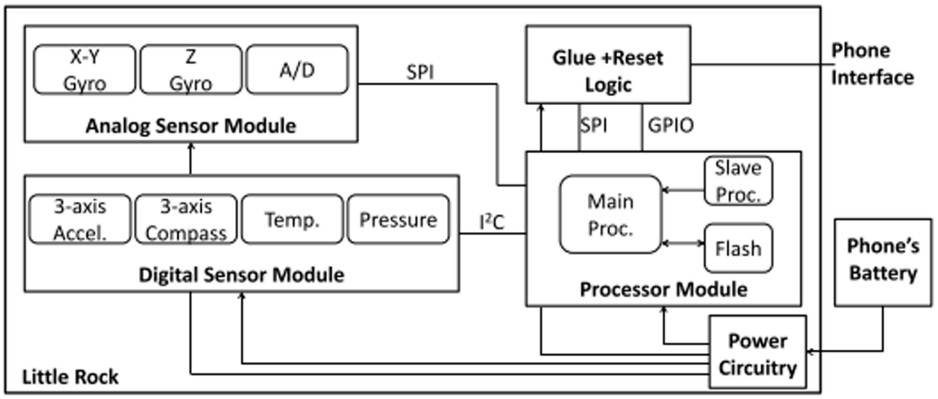


Abbildung 2.5: Überblick der Little Rock Architektur [Priyantha et al., 2010]

Ein weiteres Problem bei „Mobile Phone Sensing“ liegt in der Tatsache, dass es sich um ein mobiles Gerät handelt dessen Kontext sich laufend verändert. Dieser Umstand macht es um ein vielfaches schwerer Daten vernünftig aufzuzeichnen. Ein typischer Umstand, der auch sehr gerne als Beispiel genommen wird, ist der Ort an dem sich ein Smartphone befindet. Der Benutzer kann es in der Hosentasche mit sich herumtragen, es gerade in der Hand halten oder es einfach irgendwo auf dem Schreibtisch liegen haben. In jedem Fall werden Daten aufgezeichnet, doch sollen diese auch miteinander vergleichbar sein, obwohl sie sich in einem anderen Kontext befinden.

2.2.2 Dateninterpretierung

Nachdem nun die Daten aufgezeichnet sind muss man versuchen von ihnen zu lernen bzw. sie richtig zu interpretieren, denn die aufgezeichnete Rohdaten alleine sind meist nutzlos wenn sie nicht weiter verarbeitet werden. Zur weiteren Interpretierung können verschiedenste „Data Mining“ Verfahren oder statistische Werkzeuge genutzt werden, um die benötigten Informationen zu gewinnen. Dies wird in vielen Applikationen dafür verwendet um dem Benutzer Zusammenfassungen, über zum Beispiel seine zurückgelegte Strecke, anzuzeigen.

Ein andere Möglichkeit die Daten zu nutzen wäre das Modellieren des Nutzerverhaltens und des zugehörigen Kontextes um daraus Schlüsse zu ziehen. Um aus den aufgezeichne-

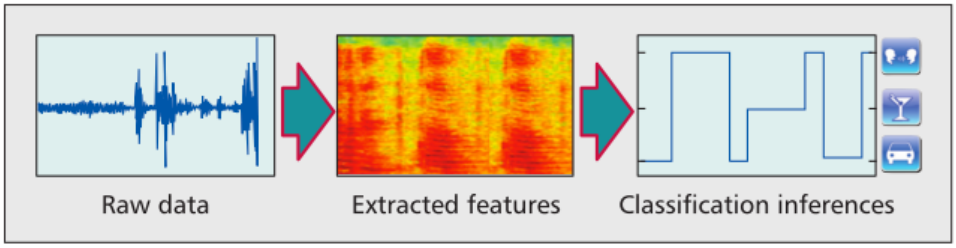


Abbildung 2.6: Ablauf des „Supervised-Learning“ Algorithmus zur Klassifikation von Audio Daten [Lane et al., 2010]

ten Daten Schlüsse ziehen zu können werden „Machine Learning“ Algorithmen verwendet. Laut [Lane et al., 2010] ist bei Verwendung auf dem Smartphone ein „Supervised-Learning“ Algorithmus am besten dafür geeignet. Dabei werden, wie in Abbildung 2.6 zu sehen ist, aus den aufgezeichneten Rohdaten ein Feature-Vektor extrahiert und diese dem Algorithmus zur Klassifikation übergegeben. Die Klassifikation ist vom zuvor festgelegten und gekennzeichneten Tranigsset abhängig. Der Vorteil dabei ist die direkte Einflussnahme auf die Klassen. Als Nachteil ist jedoch der zusätzliche Aufwand des Trainings zu sehen. Aus diesem Grund ist der „Supervised Learning“ Algorithmus vor allem für kleinere Applikation geeignet. Um mit dem Verhalten und dem Kontext von großen Communities zurecht zu kommen sollten aber „Semi-Supervised“ oder „Unsupervised-Learning“ Algorithmen verwendet werden.

2.2.3 Datenverwertung

2.2.3.1 Informieren

Die gesammelten Daten werden in den meisten Applikation dazu genutzt den Benutzer über die aufgezeichneten Daten zu informieren. Ob dies nun die Rohdaten oder bereits aggregierte Daten, so wie in Abbildung 2.7, sind spielt dabei keine Rolle. Die Visualisierung oder auch rein textuelle Darstellung der Daten ist von der jeweiligen Applikation und den zur Verfügung stehenden Daten abhängig. Doch sollte der Kreis in dem der Nutzer seine Daten aufzeichnet und verarbeiten lässt immer geschlossen werden. Dies geschieht dann wenn der Nutzer gewonnenen Informationen zurückgeliefert bzw. angezeigt bekommt.

2.2.3.2 Teilen

Es gibt verschiedene Arten seine Daten mit anderen zu teilen. Die klassische Methode dafür ist jedoch, das Hochladen der eigenen Daten in ein Web-Portal. In diesem werden dann die

Abbildung 2.7: Das animierte Aquarium mit den drei unterschiedlichen Tieren, dient als aggregierte Visualisierung von mobilen Sensorendaten und soll die verschiedenen Dimensionen des Wohlbefindens widerspiegeln. [Lane et al., 2011]

Daten für den persönlichen Gebrauch visualisiert und, meist in aggregierter Form, mit anderen geteilt. Eine weitere Möglichkeit ist es die Daten in ein bereits existierendes System einfließen zu lassen. Damit werden die bestehenden Systeme mit wertvollen Daten angereichert. Des weiteren werden die Daten weiter verbreitet und sind somit leichter zugänglich.

Ein weiterer wichtiger Punkt bezieht sich auf den Gemeinschaftsgedanken eines Menschen und bezieht somit soziale Plattformen wie Facebook und Twitter mit ein. Eine Produktgruppe die exzessiven Gebrauch davon macht sind die unzähligen Fitness-Apps. Ein Beispiel dafür wäre die beliebte Fitness-App runtastic[1]. In dieser Applikation ist es möglich seine Ergebnisse automatisch auf diversen sozialen Plattformen, wie in Abbildung 2.8 zu sehen ist, zu veröffentlichen und auf diese Weise mit seinen Freunden zu teilen.

2.2.3.3 Beeinflussen

Die aufgezeichneten Daten können nicht nur genutzt werden um den Benutzer über vergangenen Aktivitäten zu informieren oder diese Daten mit anderen zu teilen. Die aufgezeich-

[1] http://www.runtastic.com

Abbildung 2.8: Facebook Status Update nach Beendigung einer runtastic Aktivität.

neten Daten können zusammen mit den Daten einer Community dazu genutzt werden den Benutzer gezielt bei seinen Aktivitäten zu beeinflussen. Oft reicht es nicht aus den Benutzer nur mit den eigenen Daten zu motivieren. Wenn der Benutzer sich jedoch mit anderen Gleichgesinnten vergleichen kann steigert das seine Motivation enorm. [Fogg, 2002]

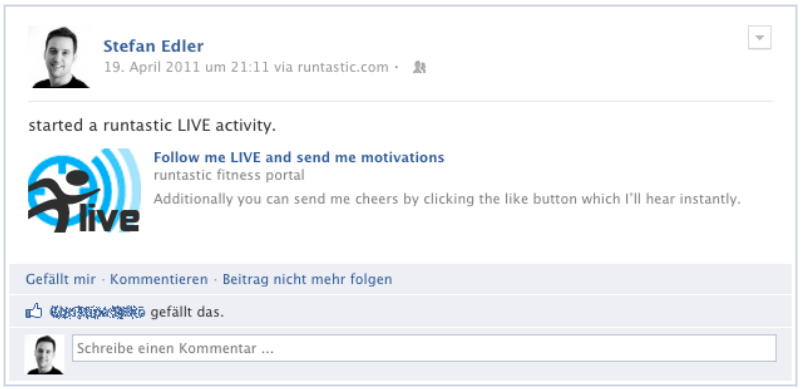


Abbildung 2.9: Facebook Status Update beim Start einer runtastic Live Aktivität.

Zusätzlich zum Gedanken des, Teilens mit Freuden, kommt bei runtastic und bei allen anderen Fitness-Applikationen der Gedanke der Motivation. Dabei wird, wenn gewünscht, die Aktivität automatisch nach Beendigung auf verschiedenen sozialen Plattformen veröffentlicht. Diese Funktion ist in vielen Applikationen bereits zum Standard geworden. runtastic geht jedoch noch einen Schritt weiter und bietet zusätzlich noch die Möglichkeit an die Aktivität Live zu veröffentlichen. Dabei wird, wie in Abbildung 2.9 zu sehen ist, die Aktivität bereits beim Start auf den sozialen Plattformen veröffentlicht. Danach kann jeder die gerade laufende Aktivität verfolgen, kommentieren und den Benutzer sogar dabei

anfeuern. Dabei stehen verschiedene Arten der Anfeuerung zur Auswahl (siehe Abbildung 2.10), welche dann direkt beim Benutzer abgespielt und auch nachträglich im Aktivitätenlog angezeigt werden.

Abbildung 2.10: Erhaltene Motivationen von Freunden über soziale Plattformen

2.3 Quantified Self

Viele Unternehmen zeichnen bereits die verschiedensten Daten über ihr eigenes Unternehmen auf, wie zum Beispiel den Gewinn jedes einzelnen Jahres bzw. Monats, oder das sich ständig verändernde Inventar. Aber auch Universitäten zeichnen die Ergebnisse aus Prüfungen und deren Beteiligungen auf. Dieses akribische Aufzeichnen von Daten war bis vor kurzem hauptsächlich im beruflichen Alltag zu finden. Jedoch wird die Anzahl der Einzelpersonen die Daten, aus den verschiedensten Gründen, aufzeichnen stetig größer. Diese Personen glauben, dass das Aufzeichnen und Analysieren ihrer Aktivitäten sie in ihrem Leben unterstützt bzw. ihr Leben verbessert. Diese Herangehensweise wird auch als „self-tracking“, „body hacking“ oder „self-quantifying“ bezeichnet [Economist, 2012]. Im Jahre

Abbildung 2.11: Quantified Self Logo

2007 wurde von den Wired Journalisten Gary Wolf und Kevin Kelly die Bewegung Quantified Self[2] ins Leben gerufen. Sie wollen damit allen Personen, die daran interessiert sind Daten über sich selbst aufzuzeichnen und zu analysieren, eine Plattform bieten um sich mit Gleichgesinnten auszutauschen, sich auf dem neuesten Stand zu halten oder sich darüber zu informieren. Seit 2001 finden sogar internationale Konferenzen mit Anwendern, Entwicklern, Unternehmen der Gesundheitsbranche und Journalisten statt.

[2] quantifiedself.com

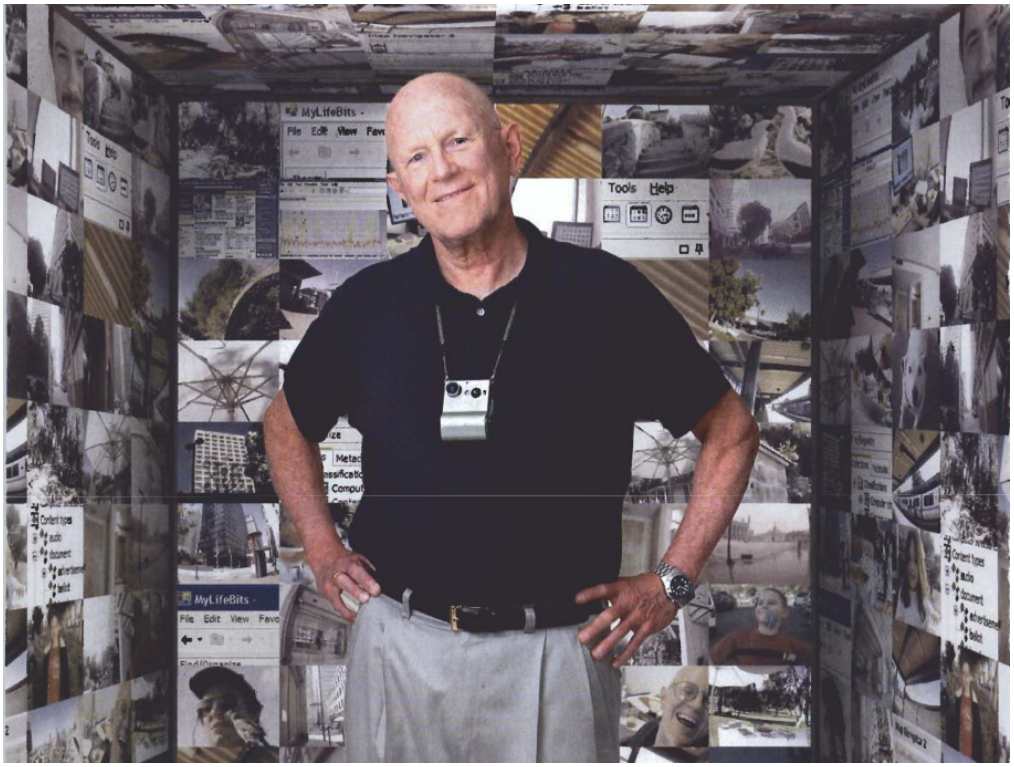


Abbildung 2.12: Gordon Bell [Cherry, 2005]

Es gab jedoch bereits, bevor diese Quantified Self Bewegung entstand, Forschungen und Entwicklungen im Bereich der Aufzeichnung und Speicherung persönlicher Daten. Der wohl bekannteste Vertreter ist Gordon Bell, Wissenschaftler bei Microsoft Research. Er ist beteiligt am Projekt MyLifeBits, welches Daten einer Person über sein gesamtes Leben aufzeichnet. Grundlage für dieses Projekt war das Memex von Vannevar Bush [Bush, 1945], das jedoch zu einer Zeit entstand in der es technologisch noch nicht möglich war all diese Daten aufzuzeichnen [Gemmell et al., 2002].

Gordon Bell selbst stellte sich für das Projekt als Versuchsperson zur Verfügung und zeichnet nun bereits seit 1998 seine persönlichen Daten auf. Zu den aufgezeichneten Daten zählen Bücher, Zeitschriften, E-Mails, besuchte Webseiten, gekaufte Lieder, Konversationen und Fotos. Zusätzlich zu den ab 1998 anfallenden Daten scannte er auch noch alle bisherigen Artikel, Bücher, Karten, Briefe, Notizen, Poster und Fotos seines privaten und beruflichen Lebens. Weiters digitalisierte er selbst aufgezeichnete Videos und Sprachaufzeichnungen. Um den Umstieg in ein Papierloses Leben zu schaffen benötigte Gordon Bell einen persönlichen Assistenten der für die Umsetzung mehrere Jahre benötigte [Bell und Gemmel, 2007].

Wie in Tabelle 2.13 zu sehen ist geht Bell davon aus, dass man pro Jahr ca. 18 Gigabyte an Daten ansammelt. Das bedeutet weiter er kann mit einer 1 Terabyte großen Festplatte, 60 Jahre seines Lebens aufzeichnen. Zu Beginn des Projektes MyLifeBits war 1 Terabyte Speicher noch recht viel und auch nur teuer zu bekommen. Doch durch die technologischen Entwicklungen ist heutzutage 1 Terabyte Speicher leicht und auch zu einem erschwinglichen Preis zu bekommen.

ITEM	SIZE EACH (BYTES)	DAILY RATE	ANNUAL STORAGE (BILLIONS OF BYTES)	60-YEAR STORAGE (BILLIONS OF BYTES)
Books/reports	1,000,000	1	0.4	21.9
E-mails	5,000	100	0.2	11.0
Scanned pages	100,000	5	0.2	11.0
Web pages	50,000	100	1.8	109.5
Purchased songs (MP3)	4,000,000	1	1.5	87.6
Recorded speech	1,000/ second	Eight hours	10.5	630.7
Photos (JPEG)	1,000,000	10	3.7	219.0
TOTAL			18.2	1,090.6

Abbildung 2.13: Übersicht über den Speicherverbrauch der gesammelten Daten [Bell und Gemmel, 2007].

Kapitel 3

Implementation

3.1 Einleitung

Dieses Kapitel behandelt die praktische Umsetzung der Masterarbeit. Dabei werden die beiden entstandenen Frameworks, „SensorKit“ und „SensorModel“, sowie deren Zusammenspiel näher erläutert. Die beiden Frameworks wurden an die zuvor vorgestellte generische Architektur angelehnt und setzen vokalem die drei Hauptkomponenten, Aufzeichnung, Interpretierung und Datenverwertung, der Architektur um. Die Frameworks wurden in Objective-C unter Verwendung des CocoTouch Frameworks umgesetzt. Sie können auf iPads sowie auf iPhones ab iOS Version 4.0 eingesetzt werden. Da Apple die Unterstützung für ARMv6 Prozessoren mit der iOS Version 6.0 einstellt, und somit jedes Gerät auf iOS Version 4.0 läuft, werden alle verfügbaren iPads und iPhones unterstützt. Bevor der eigentliche Teil der Implementierung der Masterarbeit beginnt, werden noch die Einschränkungen der mobilen Entwicklung, im speziellen die der iOS Geräte, beschrieben. Diese Einschränkungen stellen eine zusätzliche Herausforderung bei der Entwicklung für iOS Geräte dar.

3.2 Herausforderungen

In der Entwicklung von Applikation für iOS Geräte existieren diverse Einschränkungen die von jedem Entwickler bereits zum Start eines Projektes beachtet werden müssen. Diese Einschränkungen auf mobilen Geräten zählen zu den großen Herausforderungen der mobilen Entwicklung. Es gibt Einschränkungen die sich auf die Hardware und solche die sich auf die Software beziehen. Weiters sind gewisse Einschränkungen gegeben und können teilweise

nicht umgangen werden, andere wiederum wurden bewusst vom Hersteller eingebaut um, zum Beispiel die Sicherheit zu erhöhen.

Die Hardwareeinschränkungen fallen auf den verschiedenen Versionen der iOS Geräte natürlich unterschiedlich aus. Die Abbildungen 3.1 und 3.2 sollen einen Überblick über alle bereits erschienen Geräte geben. Weiters sind die unterschiedlichen Spezifikationen aufgelistet an denen man den Fortschritt der Geräte sieht. Ein schnellerer Prozessor und eine Erhöhung des Arbeitsspeichers ist bei Apple jedoch kein Garant für flüssigere Applikationen. Durch die Einführung des Retina Displays wurde zwar den Geräten mehr Performance verpasst die jedoch durch die Erhöhung der Auflösung nicht spürbar ist.

3.2.1 Speicher

iOS Geräte besitzen einen integrierten Flash-Speicher und haben keine Möglichkeit diesen, zum Beispiel über SD-Karten, zu erweitern. Der integrierte Speicher beinhaltet eine kompakte OS X Installation die nur einige hundert Megabyte groß ist. Darin sind bereits die vorkompilierten Frameworks enthalten, die von den Entwicklern genutzt werden können. Damit ist die Basis jedes Gerätes geschaffen aber damit auch der erste wichtige Speicher verbraucht. [Sadun, 2012]

Weiters ist jede Applikation in ihrer Größe auf maximal 2 GB limitiert. Außerdem können Applikation die größer als 20 MB sind ausschließlich über Wi-Fi heruntergeladen werden. Damit soll verhindert werden, dass das Downloadvolumen des Benutzers nicht unnötig belastet oder gar zu schnell überschritten wird. [Sadun, 2012]

3.2.2 Datenzugriff

In der iOS Welt läuft jede Applikation in ihrer eigenen Sandbox. Das bedeutet, sie läuft in einer festgelegten und abgeschlossenen Umgebung und kann somit nicht direkt auf andere Applikationen, Daten oder Ordner zugreifen. Es ist jedoch möglich auf Daten aus der iCloud oder der Zwischenablage, dem *UIPasteboard*, zuzugreifen. Weiters ist es möglich mithilfe des *UIDocumentInteractionController*[1] Daten mit anderen Applikation zu teilen und somit einen eingeschränkten Austausch der Daten zu ermöglichen. Es wird dabei nämlich nicht auf die selbe Resource zugegriffen, sondern diese komplett in die Sandbox der anderen Applikation kopiert. [Sadun, 2012]

[1] http://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide

3.2.3 Arbeitsspeicher

Der Arbeitsspeicher eines iOS Gerätes ist wahrscheinlich die kritischste Resource für einen Entwickler. Das besondere dabei ist, dass es auf iOS Geräten keine Möglichkeit gibt Arbeitsspeicher auf den internen Speicher zu swapen. In dem Fall, dass die Applikation zu viel Arbeitsspeicher benötigt, wird die Applikation einfach beendet. Um den Entwickler bei der Speicherverwaltung zu unterstützen wurden Memory Warning Levels[1] eingeführt auf die der Entwickler mit der Freigabe von Ressourcen reagieren sollte. [Sadun, 2012]

3.2.4 Interaktion

Die Interaktion mit iOS Geräten kann man zwar als Einschränkung ansehen da es keine Unterstützung für eine Maus gibt, jedoch gibt es viele andere Möglichkeiten der Interaktion. In den iOS Versionen bis Version 3.2 gab es außerdem noch keine Unterstützung für eine Tastatur. Doch seit Version 3.2 können Bluetooth und USB Tastaturen (keine offizielle Unterstützung und nur in Verbindung mit dem „Apple Camera Connection Kit“) zur Eingabe von Text verwendet werden. [Sadun, 2012]

Ein weitere wichtiger Punkt der Interaktion betrifft direkt das Applikations Design. Es ist in jedem Bereich darauf Rücksicht zu nehmen, dass der Bildschirm eine sehr eingeschränkte Größe aufweist. Des weiteren ist darauf zu achten, dass die Oberfläche keine Desktop typische Bestandteile, wie zum Beispiel mehrere geöffnete Fenster, hat sondern die iOS typischen Paradigmen verwendet. [Sadun, 2012]

3.2.5 Energie/Laufzeit

Wie bei allen mobilen Plattformen muss auch bei iOS Plattformen auf den Energieverbrauch geachtet werden. Für den Benutzer ist es natürlich nicht erstrebenswert seinen Akku mehrmals täglich aufzuladen und so sollte jede Applikation so wenig Energie wie nur möglich verbrauchen. So sollten zum Beispiel Ressourcen wie GPS oder Bluetooth nur eingeschalten werden wenn sie wirklich benötigt werden und nicht die gesamte Laufzeit der Applikation. [Sadun, 2012]

3.2.6 Multitasking

Apple verfolgte bis zur iOS Version 4 die Strategie, dass nur eine Applikation eines Drittherstellers zur selben Zeit laufen dürfte. Somit war es für Entwickler nicht möglich sei-

ne Applikation in den Hintergrund zu legen oder bei Beendigung durch den Benutzer im Hintergrund weiter zu laufen. Diese Eigenschaft war den von Apple entwickelten Applikationen vorbehalten. Somit muss jede Applikation, wenn sie vom Benutzer oder auch vom System beendet wird, schnellstmöglich seine noch benötigten Daten speichern, um sie nicht zu verlieren. Mit der iOS Version 4 wurde eine sehr eingeschränkte Form von Multitasking eingeführt. [Sadun, 2012]

Mit iOS Multitasking ist es nun möglich zwischen verschiedenen Arten des Verhaltens bei Beendigung zu wählen. Das Standard Verhalten von Applikation ist es einfach zu pausieren, das heißt sie werden nicht komplett beendet sondern ihr aktueller Status wird für ein erneutes Öffnen gespeichert. Dieses Verhalten garantiert jedoch nicht, dass die Applikation nicht doch noch durch zum Beispiel eine „Memory Warning“ einer anderen Applikation beendet wird. Es ist auch möglich eine ungewisse Zeit vom Betriebssystem zur Verfügung gestellt zu bekommen, um noch einen letzten Task abzuschließen oder aber einen Task für den Hintergrund zu erstellen der auch weiter läuft wenn bereits andere Applikationen gestartet wurden. Diese Tasks sind jedoch stark limitiert und zurzeit auf

- spielen von Musik,
- sammeln von Positionsdaten und
- Voice over IP (VoIP)

begrenzt. Des weiteren laufen diese Tasks nicht ständig im Hintergrund sondern werden lediglich bei eintretenden Events vom Betriebssystem benachrichtigt und können dann darauf reagieren. Zusätzlich zu den neuen Möglichkeiten ist es auch weiterhin möglich, durch eine Einstellung in der Info.plist, seine Applikation jedes mal komplett beenden zu lassen. [Sadun, 2012]

Model	RAM/Storage	Processor	Display	Features
Original iPhone (*iPhone 1,1*)	128MB 4, 8, 16GB	620MHz Samsung RISC ARM (underclocked to 412MHz)	320×480 at 163 ppi	2.0 MP camera, speaker, microphone, accelerometer, brightness sensor, proximity sensor, telephony, vibration, 802.11b/g.
iPhone 3G (*iPhone 1,2*)	128MB 8, 16GB	620MHz Samsung RISC ARM (underclocked to 412MHz)	320×480 at 163 ppi	2.0 MP camera, speaker, microphone, GPS, accelerometer, brightness sensor, proximity sensor, telephony, vibration, 802.11b/g.
iPhone 3GS (*iPhone 2,1*)	256MB 8, 16, 32GB	833MHz ARM Cortex-A8 (underclocked to 600MHz)	320×480 at 163 ppi	3.0 MP autofocus video camera, speaker, microphone, accelerometer, brightness sensor, proximity sensor, GPS, voice control, magnetometer, telephony, vibration, 802.11b/g.
iPhone 4 (*iPhone 3,x*)	512MB 16, 32GB	1GHz Apple A4	640×960 at 326 ppi	Dual cameras, including 5.0 MP autofocus back camera and front video camera, LED flash, speaker, accelerometer, brightness sensor, proximity sensor, dual noise-cancelling microphones, GPS, voice control, magnetometer, gyroscope, telephony, vibration, 802.11b/g/n.
iPhone 4S (*iPhone 4,x*)	512MB	Apple dual-core A5	640×960	Dual cameras, including 8.0 MP autofocus back camera and front video camera, LED flash, speaker, accelerometer, brightness sensor, proximity sensor, dual noise-cancelling microphones, GPS, voice control, Siri voice assistant, magnetometer, gyroscope, telephony, vibration, 802.11b/g/n.

Abbildung 3.1: Übersicht aller iPhones inklusive Spezifikationen. Grau hinterlegte iPhones werden zum aktuellen Zeitpunkt nicht mehr verkauft. [Sadun, 2012]

Model	RAM/Storage	Processor	Display	Features
iPod touch 1G (*iPod 1,1*)	128MB 8, 16, 32GB	620MHz Samsung RISC ARM (underclocked to 412MHz)	320×480 at 163 ppi	Accelerometer, brightness sensor, 802.11b/g.
iPod touch 2G (*iPod 2,1*)	128MB 8, 16, 32GB	620MHz Samsung RISC ARM (underclocked to 532MHz)	320×480 at 163 ppi	Accelerometer, brightness sensor, speaker, 802.11b/g.
iPod touch 3G (*iPod 3,1*)	256MB 32, 64GB	833MHz ARM Cortex-A8 (underclocked to 600MHz)	320×480 at 163 ppi	Accelerometer, brightness sensor, speaker, voice control, 802.11b/g.
iPod touch 4G (*iPod 4,1*)	256MB 8, 16, 32, 64GB	1GHz Apple A4	640×960 at 326 ppi (Retina)	Accelerometer, brightness sensor, gyroscope, speaker, voice control, 802.11b/g/n, dual cameras, including 960×720 back camera with digital zoom and VGA front video camera.
iPad and iPad 3G (*iPad 1,1*)	256MB 16, 32, 64GB	1GHz Apple A4	768×1024 at 132 ppi	Accelerometer, brightness sensor, speaker, microphone, magnetometer, 802.11a/b/g/n. GPS on 3G model.
iPad 2 and iPad 2 3G (*iPad 2,x*)	512MB 16, 32, 64GB	1GHz dual-core Apple A5	768×1024 at 132 ppi	Accelerometer, brightness sensor, speaker, microphone, gyro, magnetometer, 802.11a/b/g/n, dual cameras, including 960×720 back camera with digital zoom and VGA front video camera. GPS on 3G model.
iPad 3 (*iPad 3,x*)	*TBD—likely 512MB, possibly 1GB*	*TBD—likely Apple A5*	*Likely 768×1024 or possibly 1536×2048*	*TBD—this model was not yet introduced at the time this book was being written.*

Abbildung 3.2: Übersicht aller iPods und iPads inklusive Spezifikationen. Grau hinterlegte iPods werden zum aktuellen Zeitpunkt nicht mehr verkauft. [Sadun, 2012]

3.3 SensorKit

Der praktische Teil dieser Masterarbeit besteht im wesentlichen aus 2 Teilen, wobei es sich bei einem davon um das „SensorKit“ Framework handelt. Dieses Framework beinhaltet sämtliche Sensoren bzw. deren Implementierung und ist des weiteren noch dafür verantwortlich die gewonnen Daten anderen zur Verfügung zu stellen. Dafür besteht das Framework, wie in Abbildung 3.3 zu sehen ist, aus dem *SKSensorController* der die Verwaltung der Sensoren übernimmt sowie aus beliebig vielen Sensoren die allesamt von *SKSensor* abgeleitet sind. Der *SKSensorController* ist die zentrale Verwaltungsstelle im „SensorKit“ und für alle Sensoren verantwortlich. Das heißt der Controller muss die Sensoren starten, deren aufgezeichneten Daten entgegennehmen und die Sensoren wieder beenden. Der *SKSensorController* nutzt auch das „SensorModel“, auf welches in Abschnitt 3.4 eingegangen wird, um die aufgezeichneten Daten zu persistieren.

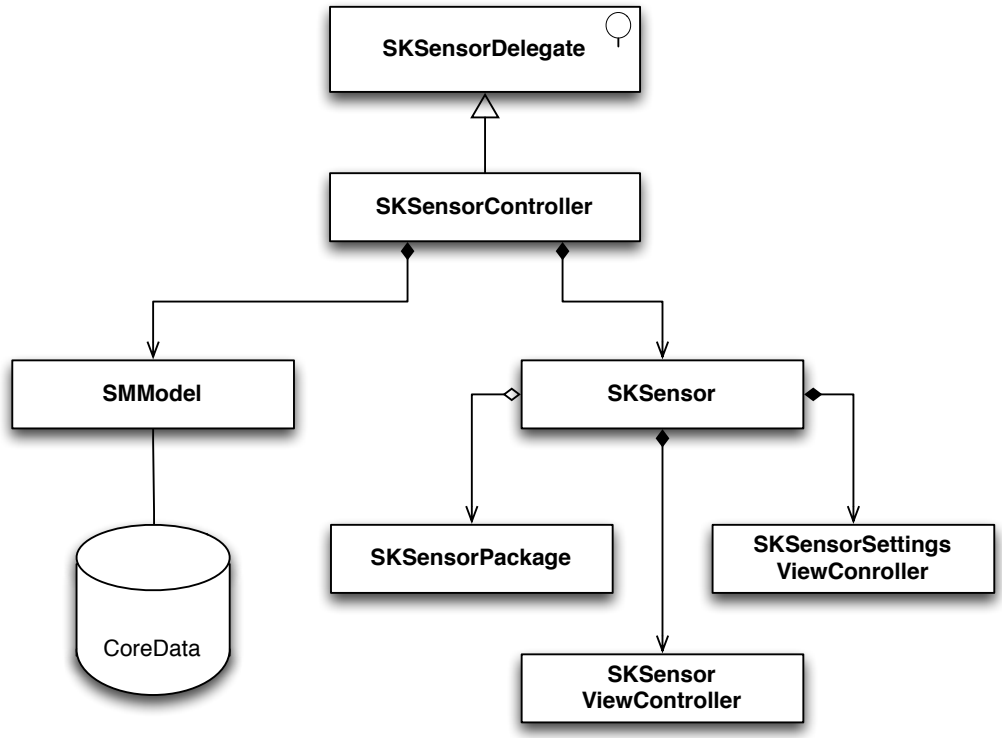


Abbildung 3.3: Klassendiagram des „SensorKit“ Framework

3.3.1 Sensoren

In diesem Framework werden drei unterschiedliche Arten von Sensoren unterschieden. Es existieren passive und aktive Sensoren, aber auch eine Kombination von beiden ist möglich. Der passive Sensor verrichtet seine Aufgaben ohne zutun des Benutzers und läuft eigenständig im Hintergrund. Anders der aktive Sensor, der nur dann Daten aufzeichnet wenn der Benutzer es wünscht bzw. anstößt. Unabhängig von der Art des Sensors werden die Sensoren, wie bereits zuvor erwähnt, immer von der Klasse *SKSensor* abgeleitet. Diese Klasse bietet, wie in Listing 3.1 zu sehen ist, eine Vielzahl an Variablen und Methoden die von den abgeleiteten Sensoren implementiert werden müssen. Der *sensorViewController* ist dafür verantwortlich die aktuellen Daten während der Aufzeichnung zu visualisieren. Bei Bedarf kann auch jeder Sensor der passiv eingesetzt werden kann optional einen *SKSensorSettingsViewController* implementieren der es ermöglicht diverse Parameter über das User Interface einzustellen. Um dem *SKSensorController* mitzuteilen um welche Art des Sensors es sich handelt, können die Sensoren beim *SKSensorController* registriert werden. Wie dieser Vorgang im Detail funktioniert ist in 3.3.2 näher beschrieben.

```
//
//  SKSensor.h
//

@interface SKSensor : NSObject {

    id<SKSensorDelegate> delegate;

    NSString*           name;
    BOOL                active;
    id                  currentValue;

    UIViewController*   sensorViewController;
}

@property (nonatomic, retain)           id<SKSensorDelegate> delegate;
@property (nonatomic, retain)           NSString* name;
@property (nonatomic, getter=isActive)  BOOL active;
@property (nonatomic, retain)           id currentValue;
@property (nonatomic, readonly)         UIViewController*
    sensorViewController;

+ (SKSensor *)sensor;

- (void)start;
- (void)stop;
- (SKSensorPackage *)sensorPackage;
- (SKSensorSettingsViewController *)sensorSettings;

@end
```

Sourcecode 3.1: SKSensor Klasse von der alle anderen Sensoren abgeleitet sind

Besondere Aufmerksamkeit sollte jedoch auf das *SKSensorPackage* gelegt werden. Diese Klasse implementiert mögliche Aggregationen und ist dafür verantwortlich die Daten für zum Beispiel Visualisierungen aufzubereiten und zur Verfügung zu stellen. Jeder Sensor muss eine Klasse implementieren die von diesem *SKSensorPackage* abgeleitet wird. Um der Visualisierung mitteilen zu könne um welchen Datentyp es sich handelt muss diese Klasse seine möglichen Kalkulationen und den daraus resultierenden Datentyp bekannt geben können. Als Beispiel dafür soll das SKMoodSensorPackage dienen welches in Listing 3.2 abgebildet ist. Dieses Beispiel implementiert ausschließlich die *init* Methode die zwingend von jeder Klasse überschrieben werden muss. In den *calculations* werden nun alle möglichen Kalkulationen und deren Datentyp angegeben. Da diese Klasse ausschließlich aus der *init* Methode besteht, bietet sie auch keine speziellen Kalkulationen sondern nutzt nur die Standard Methoden der *super* Klasse.

```
//
//  SKMoodSensorPackage.m
//

@implementation SKMoodSensorPackage

- (id)init {
    self = [super init];

    if ( self ) {
        self.calculations = [NSDictionary dictionaryWithObjectsAndKeys:
                             [NSNumber numberWithInt:
    SensorPackageDataTypeMood], @"Mood",
                             nil];
    }
    return self;
}

@end
```

Sourcecode 3.2: Implementierung des SKMoodSensorPackage

3.3.1.1 Koordinaten

Mit dem *SKLocationSensor* ist es möglich die Koordinaten des Benutzers aufzuzeichnen. Dabei werden die Positionen aber nicht in einem vordefinierten Zeitinterval an den Controller geschickt sondern nur bei einer relevanten Änderung der Position. Wie bei jedem GPS ist es auch hier möglich sehr ungenaue Positionen, aufgrund von zu wenigen Satelliten zu erhalten. Aus diesem Grund wurde versucht diese teilweise auftretenden Ausreißer zu erkennen und, um die Daten zu verbessern, zu ignorieren.

Der Sensor zeichnet zusätzlich zu den Positionsdaten des Benutzers auch noch folgende Daten, die dann gesammelt als *CLLocation* Objekt an den Controller gesendet werden, auf:

- Koordinaten (Latitude, Longitude)
- Höhenangabe (Altitude)
- Horizontale Genauigkeit
- Vertikale Genauigkeit
- Zeitstempel
- Geschwindigkeit
- Himmelsrichtung

3.3.1.2 Lautstärke

Der Lautstärkensensor nimmt zu jedem Zeitpunkt die aktuelle Umgebungslautstärke auf. Damit ist es möglich Lautstärkeninformationen über die Umgebung zu sammeln ohne die Privatsphäre der umgebenden Personen zu verletzen. Trotz des Verlustes der gesprochenen Wörter kann man aus der Lautstärke einer Kommunikation Rückschlüsse daraus ziehen. Da es beim *SKSoundSensor* zu Ausreißern, durch zum Beispiel kurzfristige Bewegung des iPhones, kommen kann wurde ein Tiefpassfilter zur Minimierung der Fehler eingebaut. Als Rückgabewert wurde ein Float gewählt der sich von -160 db bis 0 db erstrecken kann.

3.3.1.3 Kamera

Die Kamera ist ein sehr mächtiger aber auch schwer zu handhabender Sensor. Die gesammelten Bilder enthalten für einen Menschen eine Flut an nützlichen Informationen. Diese Informationen sind jedoch sehr schwer aus einem Bild zu extrahieren und in eine für einen Computer verständliche Informationen überzuführen.

Der Kamerasensor des *SensorKits* ist sowohl als passiver als auch als aktiver Sensor verfügbar und kann somit in Zeitintervalen aber auch per Benutzerinteraktion aktiv werden und dadurch Bilder aufnehmen (siehe Abbildung 3.4). Dieser Sensor verfügt im passiven Modus auch über die Möglichkeit spezielle Einstellungen vorzunehmen. Dadurch ist es möglich den Zeitinterval beliebig einzustellen aber auch die Qualität der Aufnahme und die Stärke der Kompression. Als aktiver Sensor bietet er eine iOS typische Kamera Vorschau die lediglich zum Fokussieren und Auslösen dient.

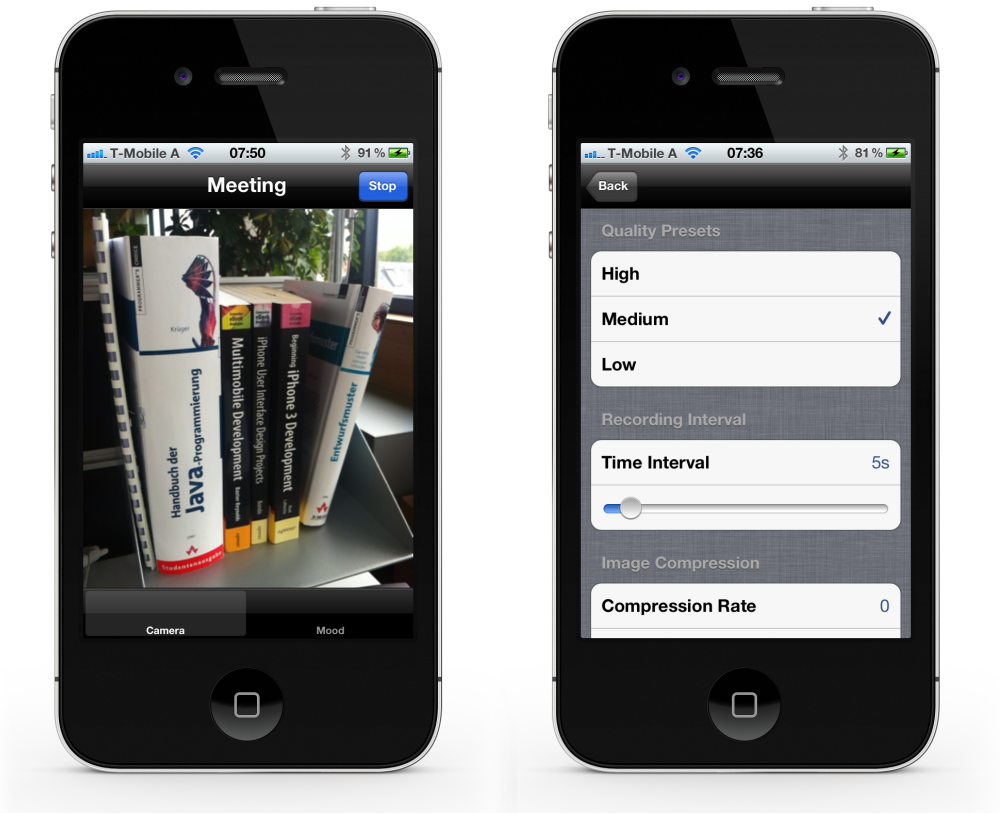


Abbildung 3.4: Kamera Sensor als aktiver Sensor mit Vorschau für den Benutzer und als passiver Sensor mit den unterschiedlichen Einstellungen

3.3.1.4 Aktivitäten

Jedes iOS Gerät verfügt über einen integrierten Beschleunigungssensor. Im *SensorKit* wurden die Daten des Beschleunigungssensors, der nur die Beschleunigung der x-, y- und z-Achse zurückgibt, nicht direkt gespeichert. Vielmehr wurden diese Daten genutzt, um die aktuelle Aktivität des Benutzers festzustellen. Diese Implementierung stellt nur einen ersten sehr einfachen Versuch dar die vom Sensor aufgezeichneten Daten zu aggregieren. Der Sensor überprüft bei jeder Veränderung der Beschleunigung in welcher Lage sich das Smartphone befindet. Befindet es sich in einer waagrechten oder einer senkrechten Lage kann man davon ausgehen, dass der Benutzer gerade sitzt oder eben steht. Wenn der Benutzer geht ändert das Smartphone ständig seine Beschleunigung und daraus kann man mithilfe von Schwellwerten eine Gehbewegung erkennen. Um korrekte Ergebnisse erzielen zu können ist es notwendig, dass Smartphone in der Hosentasche zu tragen. Einmal gestartet kann der Sensor die Aktivitäten

- Sitzen,

- Stehen und
- Gehen

erkennen und mit dem Zeitstempel, an dem die Aktivität beginnt, speichern.

3.3.1.5 Moodmap

Der Moodmap Sensor, zu sehen in Abbildung 5.3, ist ein rein aktiver vom Benutzer gesteuerter Sensor. Er wird dafür genutzt um dem Benutzer die Möglichkeit zu geben seine Gefühle während der Aufzeichnung abzuspeichern. Es ist nicht möglich sich im Nachhinein an alle Empfindungen zu erinnern bzw. diese den einzelnen Ereignissen zuzuordnen. Mit dieser Moodmap soll der Benutzer dabei unterstützt werden diese sehr wertvollen Informationen zu sammeln und später den anderen Daten gegenüber zu stellen.

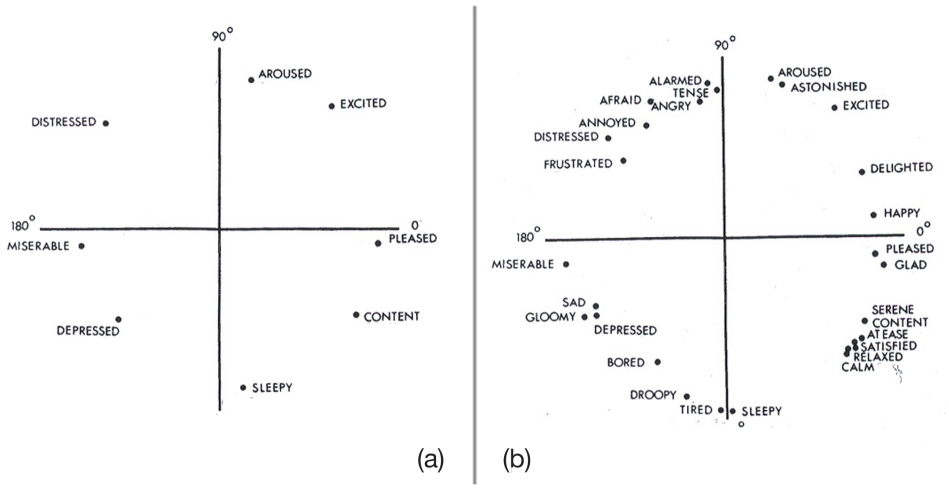


Abbildung 3.5: Circumplex Model von [Russel, 1980]: (a) Unterteilung in 8 Gefühle (b) Unterteilung in 28 Gefühle Quelle:[Russel, 1980]

Die Moodmap des *SensorKits* beruht auf dem Circumplex Model, zu sehen in Abbildung 3.5, von [Russel, 1980]. Diese Model teilt einen Kreis anhand von Gefühlen und 8 bzw. 28 unterschiedliche Bereiche. In dieser Masterarbeit wurde nicht die Ursprungsversion von [Russel, 1980] verwendet sondern die, um Farben erweiterte, Moodmap von [Stahl et al., 2005]. Der Sensor zeichnet bei jeder Berührung der Visualisierung die normierten x- und y-Koordinaten zusammen mit dem Zeitstempel auf und markiert immer den zuletzt gewählten Punkt. Die normierten Daten können dann auf jede beliebig große Moodmap übertragen werden und zusammen mit dem Zeitstempel mit anderen Ereignissen abgeglichen werden. [Fessl et al., 2012]

3.3.1.6 Wetter

Der Wettersensor zeichnet detaillierte Wetterdaten in einem definierten Zeitinterval auf. Damit der aktuelle Aufenthaltsorte nicht eingegeben und ständig aktualisiert werden muss, nutzt der Sensor zusätzlich noch den GPS-Sensor. Um den hohen Energiebedarf des GPS-Sensors möglichst niedrig zu halten, wurde die Genauigkeit des Sensors auf ungefähr 3 Kilometer festlegt. Mithilfe der erhaltenen Koordinaten wird der aktuelle Standort und dessen Wetterdaten vom „Yahoo Wetter“ Service abgerufen. Diese Daten, welche in einem XML-Format vorliegen, werden eingelesen und in einem *Weather* Objekt (siehe Listing 3.3) gespeichert und an den Controller übergeben.

```
//
//  Weather.h
//

@interface Weather : NSObject <NSCoding> {

    int             woeid;          // location id
    int             woetype;
    BOOL            updated;
    NSString*       lastUpdateOfWeatherData;

    // Location
    NSString*       street;
    int             streetNumber;
    int             postal;
    NSString*       city;
    NSString*       region;
    NSString*       state;
    NSString*       country;

    // Condition
    NSString*       conditionText;  // human readable text
    int             conditionCode;  // 0 - 47; 3200 = N/A
    int             temperature;    // in degrees

    // Wind
    int             chill;          // in degrees
    int             direction;      // in degrees
    int             speed;          // in km/h

    // Atmosphere
    int             humidity;       // in percent
    float           visibility;     // in kilometers * 100 (1400 == 14km)
    float           pressure;       // in millibars
    int             rising;         // steady: 0; rising: 1; falling: 2

    // Astronomy
    NSString*       sunrise;        // timestamp ("h:mm am/pm")
    NSString*       sunset;         // timestamp ("h:mm am/pm")

    // Units
    NSString*       temperatureUnit;
    NSString*       distanceUnit;
    NSString*       pressureUnit;
    NSString*       speedUnit;
```

```
    // Image
    UIImage*        image;
    int             width;
    int             height;
}
```

Sourcecode 3.3: Interface der Weather Klasse mit ihren Instanz Variablen

3.3.1.7 RSS-Feeds

RSS-Feeds können eigentlich nicht direkt als Sensor bezeichnet werden, jedoch ist es damit möglich unzählige zeitbezogene Informationen über einen definierten Standard abzufragen. Diese Informationen können genutzt werden um seine eigenen aufgezeichneten Daten damit anzureichern und somit detailliertere Daten zu erhalten.

Der RSS-Feed-Sensor des *SensorKits* wurde sehr generisch gehalten, somit ist es möglich beliebige RSS-Feeds hinzuzufügen und diese abzufragen. Jeder empfangene Feed, der in einem XML-Format vorliegt, wird in ein *SKEvent* Objekt umgewandelt um ihn abspeichern und später visualisieren zu können. Die *SKEvent* Klasse, zu sehen in SourceCode 3.4, beinhaltet einen Großteil der spezifizierten RSS-Daten. Zusätzlich wird noch der *SKEventType* gespeichert, um später unterscheiden zu können von welchem Sensor die Daten geliefert wurden.

```
//
//  SKEvent.h
//

@interface SKEvent : NSObject <NSCoding> {

    SKEventType     eventType;
    NSString*       eventSubType;
    NSString*       publisherName;
    UIImage*        profileImage;
    NSString*       title;
    NSString*       text;
    NSDate*         timestamp;
    NSTimeInterval  duration;
}
```

Sourcecode 3.4: Interface der SKEvent Klasse

3.3.1.8 Twitter

Twitter, als wohl der bekannteste Vertreter des Microblogging, ist ein sehr mächtiges Tool für die unterschiedlichsten Dinge. Es ist angeblich sogar möglich, anhand der Tweets den Dow-Jones Index, mit einer Genauigkeit von 86.7 %, vorherzusagen [Bollen et al., 2011]. Es ist damit möglich, ähnlich den RSS-Feeds, unzählige und sehr aktuelle Informationen zu sammeln. Der Twittersensor nutzt zur Authentifzierung das sehr beliebte oAuth Protokoll laut Hammer-Lahav [2010]. Damit ist es nur einmal nötig sich anzumelden und die Applikation freizugeben. Danach ist es möglich jederzeit ohne erneutes anmelden die Timeline des angemeldeten Benutzers abzufragen. Die Daten werden ebenfalls wie beim RSS-Feed-Sensor in einem *SKEvent* Objekt, siehe Source 3.4, zur späteren Verwendung abgespeichert.

3.3.2 Registrierung der Sensoren

Wie bereits zuvor beschrieben muss sich jeder Sensor beim Programmstart im *SKSensorController* registrieren, um später gefunden und auch gestartet werden zu können. Dafür bietet der *SKSensorController*, wie in Listing 3.5 zu sehen ist, zwei Methoden an. Die passiven Sensoren werden vor Beginn der Aufnahme ausgewählt und dann automatisch gestartet. Alle so gestarteten Sensoren laufen danach passiv im Hintergrund, das heißt sie benötigen keinen weiteren Eingaben oder Interaktionen vom Benutzer. Die aktiven Sensoren verhalten sich genau anders herum, das heißt sie werden immer gestartet, nehmen aber keine Daten automatisch auf. Sie benötigen zwingend Eingaben oder Interaktionen vom Benutzer, um genau diese Interaktionen aufzuzeichnen.

```
//
//  SKSensorController.h
//

@interface SKSensorController

- (void)registerPassiveSensor:(SKSensor *)sensor;
- (void)registerActiveSensor:(SKSensor *)sensor;

@end
```

Sourcecode 3.5: Methoden zum Registrieren der Sensoren

Wie bereits zuvor beschrieben schließen sich diese beiden Methoden nicht gegenseitig aus. Somit kann jeder Sensor sowohl als passiver als auch als aktiver Sensor tätig sein. Als Beispiel dafür dient der Kamera Sensor. Dieser kann als passiver Sensor in einem vordefi-

nierten Zeitinterval völlig automatisch Aufnahmen machen. Wenn der Zeitinterval nun aber zum Beispiel auf 30 Sekunden eingestellt wurde und der Benutzer möchte genau diesen Augenblick festhalten, ist dies möglich, da der Kamera Sensor ebenfalls als aktiver Sensor verfügbar ist. Der Benutzer kann somit, ähnlich der iOS Kammera Applikation, mit einem Tap Fotos machen.

3.3.3 Protokolle

Die einzelnen Sensoren werden, wie bereits beschrieben, vom *SKSensorController* gestartet und beginnen damit ihre Daten aufzuzeichnen. Um die unterschiedlichen Daten eines jeden Sensors an den *SKSensorController* weiterzuleiten wurde das *Delegate Design Pattern*[2] verwendet. Der *SKSensorController* setzt sich bei jedem Sensor als sein Delegate und implementiert das *SKSensorDelegate* Protokoll aus Listing 3.6. Damit kann nun jeder Sensor die Methoden des Protokolls bei seinem Delegate aufrufen.

```
//
//  SensorProtocol.h
//

@protocol SKSensorDelegate

- (void)sensor:(SKSensor *)sensor didAcquireData:(id)data;
- (void)sensor:(SKSensor *)sensor didAcquireData:(id)data atTimestamp:(NS
    Date *)timestamp;
- (void)sensor:(SKSensor *)sensor willOpenWebViewWithRequest:(NS
    URLRequest *)request;

@end
```

Sourcecode 3.6: Delegate Methoden die von den Sensoren aufgerufen werden

3.4 SensorModel

Das „SensorModel“ ist der zweite wichtige Teil dieser Masterarbeit und dafür verantwortlich die unterschiedlichen Daten der einzelnen Sensoren zu speichern. Dafür wurde das unter iOS und Mac OS X verfügbare Framework CoreData verwendet. Damit ist es möglich im Code direkt mit Objekten zu arbeiten und diese dann in die Datenbank zu speichern. Das Umwandeln der Objekte, in entsprechende Datenbankzeilen der SQLite Datenbank,

[2] http://http://developer.apple.com/library/ios/documentation/cocoa/conceptual/cocoafundamentals

wird dann vollautomatisch von CoreData[3] übernommen.

Wie im „SensorKit“ spielt auch hier der *SKSensorController* eine sehr wichtige Rolle. Dieser Controller implementiert alle Methoden die dafür verantwortlich sind Daten in die Datenbank zu schreiben, welche zu löschen oder sie abzufragen. Somit wurde ein zentraler Punkt geschaffen über den die Datenbankoperation laufen und verhindert, dass jeder Sensor sich selbst darum kümmern muss. Das hätte nur dazu geführt, dass mehr Fehler entstanden wären und diese viel schwerer zu finden wären.

3.4.1 Datenbankmodell

Das Datenbankmodell in Abbildung 3.6 verfügt über alle nötigen Entitäten, um die Sensordaten vernünftig speichern zu können. Beim Erstellen des Datenbankmodells wurde speziell auf die Leistungseinschränkungen der iOS Geräte eingegangen. Durch diese spezielle Struktur ist es möglich trotz einer enormen Menge an Sensordaten, die Datenbankabfragen sehr effizient durchzuführen.

3.4.1.1 SMSession

Die aufgezeichneten Daten werden nicht lose in die Datenbank gespeichert sondern stets an eine *SMSession*, die immer einen Anfangs- und Endzeitpunkt hat, angehängt. Das hat den großen Vorteil, dass die Abfragen, in welchen Zeitraum die Sensordaten fallen, nicht über den gesamten Datensatz erfolgen müssen. In diesem Fall muss nur nach einer passenden *SMSession* gesucht werden und über die *sensors* Beziehung gelangt man nun weiter zu den erforderlichen Daten.

3.4.1.2 SMSensor

Die *SMSensor* Entität versteht sich als eine weitere Zusammenfassung von Sensordaten. Es existiert zu jedem Sensor eine passende Entität die von der *SMSensor* Entität abgeleitet ist. Diese haben als Attribute nur den Namen des Sensors sowie den Anfangs- und Endzeitpunkt. Um nun zu den tatsächlichen Sensordaten zu gelangen muss man lediglich die *records* Beziehung abfragen.

[3] http://http:/developer.apple.com/library/ios/documentation/cocoa/conceptual/coredata

3.4.1.3 SMRecord

Diese Entität entspricht nun einer einzelnen Sensoraufzeichnung. In ihr wird der genaue Zeitstempel der Aufzeichnung sowie der zugehörige Wert im Attribut *value* gespeichert. Um in diesem Attribut alle Arten von Sensordaten speichern zu können wurde dem Attribut als Datentyp *id* zugewiesen. Damit ist es nun möglich jedes Objekt, dass das *NSCoding* Protokoll[4] implementiert in dieser Entität zu speichern.

3.4.1.4 SMProfile

Obwohl in der aktuellen Implementierung von einem einzelnen Benutzer ausgegangen wird wurde bereits die *SMProfile* Entität hinzugefügt. Damit wäre es möglich auf dem selben iOS Gerät mehrere Benutzer zu verwalten. Wie in Kapitel 5 beschrieben gibt es in der Prototyp Implementierung auch eine Möglichkeit seine Daten anderen zu schicken. Mithilfe des *SMProfile* wäre es nun möglich seine Daten von denen eines anderen zu unterscheiden.

[4] http:// developer.apple.com/library/ios/documentation/Cocoa/Conceptual/Archiving/Archiving.pdf

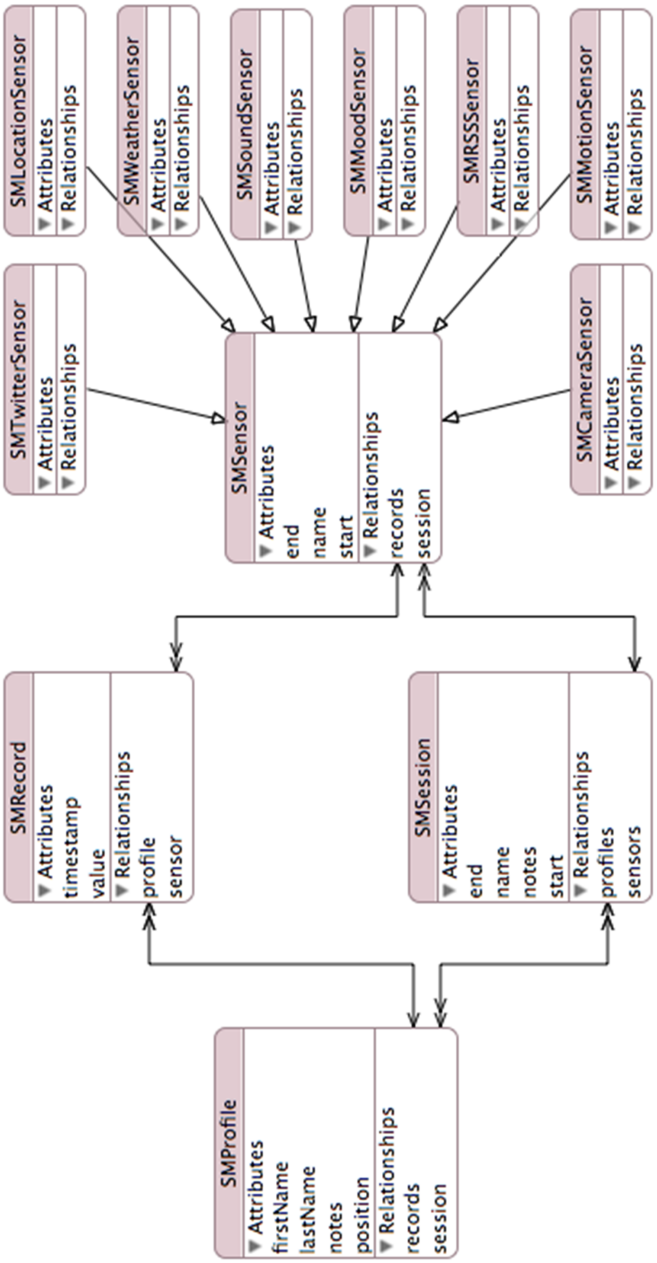


Abbildung 3.6: CoreData Datenbankmodell des SensorModels

Kapitel 4

Ähnliche Arbeiten

4.1 Jigsaw Continous Sensing Engine

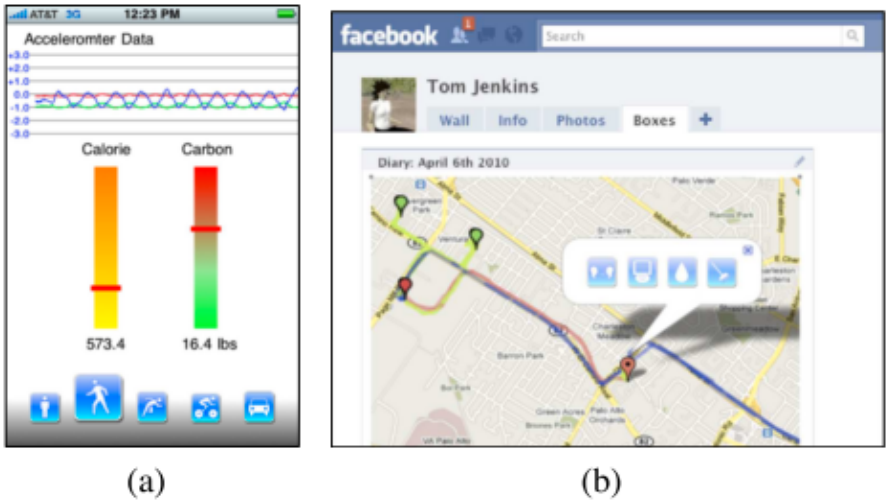
Verwendete Sensoren:

- *Beschleunigunssensor*
- *GPS*
- *Mikrofon*

Das Jigsaw Projekt des Dartmouth College versucht eine Langzeitaufzeichnung von menschlichen Aktivitäten und Kontext auf mobilen Geräten zu ermöglichen. Dabei wurde ein Framework entwickelt welches zurzeit Daten des Beschleunigungssensors, des GPS-Sensors und des Mikrofons berücksichtigt. Jeder Sensor füttert dabei eine zugehörige Pipeline, die auf die entsprechenden Bedürfnisse jedes Sensors angepasst ist, mit ihren Rohdaten. Diese werden dann entsprechend dieser Pipeline direkt auf dem Gerät verarbeitet und zurückgeliefert. Um zukünftigen Entwicklungen der Hardwarehersteller gerecht zu werden wurde bereits in der Design-Phase des Projektes darauf eingegangen. Somit ist es den Entwicklern möglich das Framework um beliebige interne Sensoren zukünftiger Geräte zu erweitern. [Lu et al., 2010]

Um die Fähigkeiten von Jigsaw demonstrieren zu können wurden die zwei Proof-of-Concept Apps JigMe und GreenSaw entwickelt. Erstere könnte man eher in die Kategorie der sozialen Netzwerke einordnen und GreenSaw in die Kategorien Gesundheit und Umwelt. [Lu et al., 2010]

Bei JigMe handelt es sich um eine opportunistische Anwendung die automatisch den Tagesablauf seines Benutzers aufzeichnet. Dabei erstellen die drei Sensoren, die durchgehend im Hintergrund laufen müssen, ein Protokoll mit Aktivitäten und GPS-Daten. Dieses Protokoll wird dann an Facebook gesendet wo es mithilfe einer Google-Map angezeigt wird. Dabei werden zusätzlich Orte an denen der Benutzer länger verweilte mit Tags ge-



(a) (b)

Abbildung 4.1: Proof-of-Concept Apps um die Jigsaw Engine zu demonstrieren. Die (a) GreenSaw App zeigt den aktuellen Kalorienverbrauch und CO2 Ausstoß. Bei der (b) JigMe App werden die Aktivitäten, Fortbewegungsarten und signifikanten Orte eines Nutzers aufgezeichnet und später in einer Map visualisiert. [Lu et al., 2010]

kennzeichnet. Ein grüner Tag bedeutet eine kürzeren Aufenthalt und ein roter Tag einen längeren. Beim Klicken auf einen Tag werden in einem Pop-up Fenster, wie in Abbildung 4.1(b) zu sehen ist, alle erkannten Aktivitäten angezeigt sowie die Tonaufzeichnung dieses Ortes abgespielt. So wie die Tags farblich gekennzeichnet werden so wird auch der zurückgelegte Weg je nach Fortbewegungsart farblich markiert, um auf einen Blick erkennen zu können wie man von einem zum andren Ort gelangt ist. Lu et al. [2010]

GreenSaw hingegen gibt dem Benutzer, wie in Abbildung 4.1(a) zu sehen ist, einen Überblick über seinen täglichen Kalorienverbrauch und CO2-Ausstoss. Um diese Informationen zu generieren werden lediglich das Geschlecht, das Gewicht, die Größe und das Automodell benötigt.Lu et al. [2010]

4.2 CenceMe

Laut [Miluzzo et al., 2008] gehören „where r u?“ und „what u doing?“ zu den am häufigsten verschickten Textnachrichten. Mit den aktuellen Smartphones, die meist eine Fülle an Sensoren beinhalten, ist es möglich diese Fragen völlig automatisch zu beantworten. Die von den Sensoren gesam-

Verwendete Sensoren:

- *Mikrofon*
- *Kamera*
- *GPS*
- *Bluetooth*

melten Daten werden dazu genutzt die aktuelle Aktivität des
Benutzers zu bestimmen. Diese Informationen werden dann über soziale Netzwerke, wie Facebook, mit anderen geteilt.

CenceMe wurde auf einem Nokia N95 implementiert und nutzt dort einen Großteil der verfügbaren Sensoren. Wie in Abbildung 4.2 zu sehen ist werden die gesammelten Rohdaten gespeichert bzw. mithilfe von Classifiern zu *Primitives* weiter verarbeitet. Um die Integrität der Daten und die Privatsphäre des Benutzers zu wahren werden diese Daten jedoch spätestens nach dem Upload vom Gerät gelöscht. Der Upload an einen Server ist nötig, da auf dem Smartphone nur eine erste Klassifizierung der Daten durchgeführt wird. Bevor die Daten in einem sozialen Netzwerk veröffentlicht werden können müssen sie zuvor noch auf dem Server eine weitere Klassifizierung durchgeführt werden. [Miluzzo et al., 2008] [Miluzzo et al., 2007]

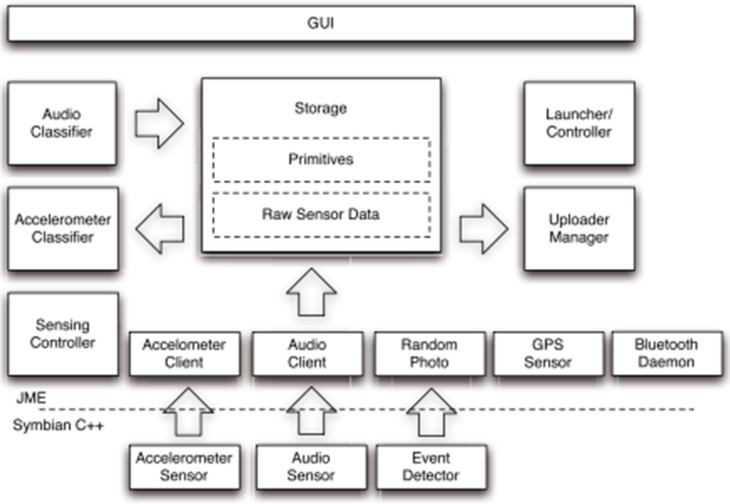


Abbildung 4.2: Miluzzo et al. [2008]

4.3 UbiFit Garden

Das „UbiFit Garden“ System nutzt körpernahe Sensoren, in Echtzeit agierende statistische Modelle und ein mobiles Anzeigegerät, um den Nutzer bei seinen körperlichen Aktivitäten zu animieren. Dabei wurde das System für den persönlichen Gebrauch einer einzelnen Person entwickelt, die sich der Notwendigkeit körperlicher Aktivität bewusst ist dieser aber nicht bzw. nicht regelmäßig nachgegangen ist.

Verwendete Sensoren:

- *Beschleunigunssensor*
- *Barometer*

Das System setzt sich aus den drei unterschiedlichen Komponenten, Fitness Gerät, interaktive Applikation und visuelle Anzeige zusammen die im folgenden näher erklärt werden.

4.3.1 Fitness Gerät

„UbiFit Garden“ ist Teil eines größeren Forschungsprojektes, dass sich mit Sensoren und Rückschlüssen auf Aktivitäten beschäftigt. Aus diesem Grund basiert das System auch auf der „Mobile Sensing Platform“ (MSP) einem System zur Erkennung von Aktivitäten. Bei der MSP handelt es sich um einen kleinen batteriebetriebenen Computer mit unterschiedlichsten Sensoren, um eine große Bandbreite an Applikationen zu unterstützen. Choudhury et al. [2008] Im „UbiFit Garden“ System wird die MSP benutzt um in Echtzeit automatisch auf körperliche Aktivitäten schließen zu können. Dafür werden die Daten auf der MSP klassifiziert und in die Kategorien Gehen, Laufen, Radfahren, benutzen eines Cross-Trainers oder Steppers eingeteilt. Diese Daten werden dann über Bluetooth an ein mobiles Gerät übertragen auf dem die interaktive Applikation und die visuelle Anzeige laufen.

4.3.2 Interaktive Applikation

Die interaktive Applikation wurde mit dem MyExperience Framework speziell für Mobiltelefone entwickelt. Sie enthält Detailinformationen über die erkannten Aktivitäten, ein Journal (siehe Abbildung4.3 a)) in dem es möglich ist Informationen über Aktivitäten hinzuzufügen, zu editieren oder auch zu löschen und eine Liste der Ziele mit dem aktuellen Fortschritt (siehe Abbildung4.3 b)). Weiters wird beim Benutzer nach längerer Inaktivität nachgefragt, ob er irgendetwas hat das er hinzufügen möchte. Diese Applikation ist auch dafür verantwortlich die visuelle Anzeige, die als nächstes behandelt wird, auf dem neuesten Stand zu halten.

4.3.3 Visuelle Anzeige

Die visuelle Anzeige beschränkt sich ausschließlich auf Grafiken um Aktivitäten und die Erreichung von Zielen darzustellen. Um den Benutzer ständig an seine Aktivitäten und Ziele zu erinnern wurde diese Anzeige als Hintergrundbild des Mobiltelefons eingerichtet. Als Metapher wurde ein blühender Garten gewählt der mit zunehmenden Aktivitäten immer weiter wächst. Die weißen Schmetterlinge visualisieren das Erreichen von kürzlich erreichten Zielen und der große gelbe Schmetterling das Erreichen des Wochenziels. In [Consolvo

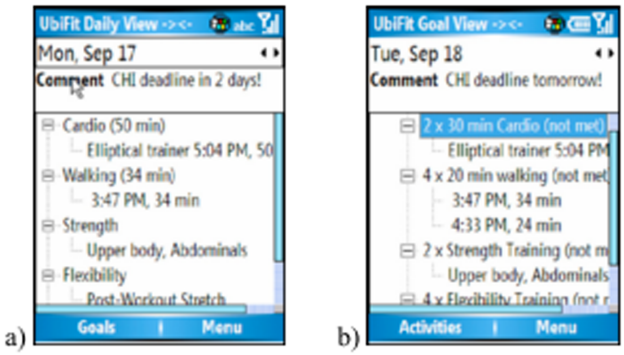


Abbildung 4.3: a) Liste der Aktivitäten eines Tages, b) Anzeige der Ziele bzw. des Fortschritts Consolvo et al. [2008]

et al., 2008] wurde herausgefunden, das Visualisierungen über das Hintergrundbild eines Smartphones sehr effektiv sind und somit einen besonderen Mehrwert bieten.

Abbildung 4.4: a) Darstellung zu Beginn der Woche b)Anzeige von unterschiedlichen Aktivitäten c) Darstellung auf einem mobilen Gerät. Consolvo et al. [2008]

4.4 SoundSense

Bei SoundSense handelt es sich um ein Framework das sich ausschließlich mit dem Mikrofon als Sensor beschäftigt und versucht diesen als persönlichen Ereignissensor zu verwenden. Geräusche die mit einem Mikrofon ausgezeich-

Verwendete Sensoren:

- *Mikrofon*

net werden beinhalten eine Vielzahl an Informationen. Diese können genutzt werden um Schlussfolgerungen über eine Person, deren Umgebung oder sozialen Ereignisse ziehen zu können. Einige Möglichkeiten des Sensors sind die Erkennung von Gesprächen, verschiedenen Aktivitäten und sozialen Strukturen einer Person aber auch die Klassifikation zu einem Ort und zu einem Ernährungsverhalten. Lu et al. [2009]

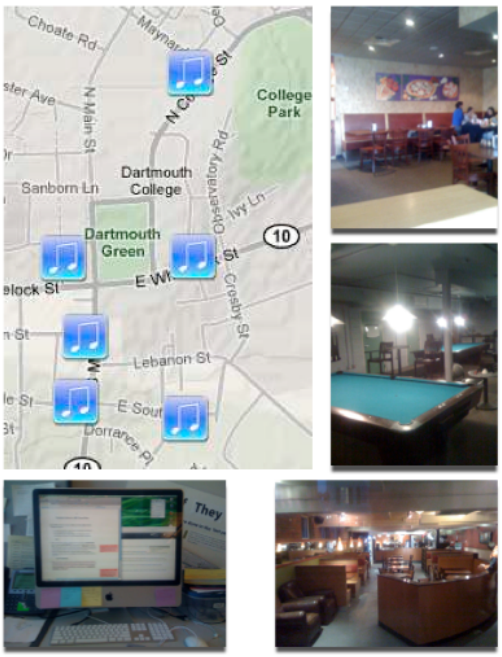


Abbildung 4.5: Screenshot der Applikation mit Orten und den zugehörigen Bildern. Lu et al. [2009]

Mit SoundSense wird ein skalierbares Framework zur Erkennung von Geräuschen für mobile Geräte zur Verfügung gestellt. Dabei wird besonderen Wert auf die unterschiedlichen Herausforderungen, die mobile Geräte mit sich bringen, eingegangen:

- Skaliert auf eine große Anzahl von Personen, wovon jede einzelne eine andere Geräuschumgebung haben kann.
- Funktioniert robust in verschiedensten Gegebenheiten also nicht nur wenn das Mobiltelefon am Tisch liegt, sondern auch wenn es in der Hosentasche getragen wird und dabei Störgeräusche auftreten.
- Die Algorithmen müssen direkt auf dem Mobiltelefon laufen dürfen aber nicht dessen Funktionalität einschränken.
- Privatsphäre des Benutzers muss in jedem Fall gewahrt bleiben.

4.5 BeWell

Verwendete Sensoren:

- *GPS*
- *Beschleunigungssensor*
- *Mikrofon*

Laut [Lane et al., 2011] ist es eine große Herausforderungen, Technologien zu entwickeln, die den Benutzer dabei unterstützen ein gesundes Leben zu führen. Erst durch die Einführung von Smartphones und deren integrierten Sensoren wurde es möglich mobile Applikation zu entwickeln, die das Wohlbefinden einer Person überwachen, abbilden und sie sogar dabei unterstützt. Mit der BeWell Applikation ist es nun möglich Aktivitäten, wie Schlafen, Bewegung und soziale Interaktion zu erkennen und aufzuzeichnen. Die gesammelten Informationen werden dann genutzt, um den Benutzer intelligentes Feedback zu geben und ihn bei einem gesünderen Leben zu unterstützen.

In Abbildung 4.6 kann man eine grobe Architektur des gesamten Systems sehen. Diese Architektur wird nun im folgenden näher beschrieben.

4.5.1 Sensor Service

Der Sensor Service, der ausschließlich im Hintergrund arbeitetet, besteht aus einer, selbst entwickelten Platform unabhängigen, C Library und Geräte spezifischen Komponenten die in Java geschrieben wurden. Bei der C Library handelt es sich um eine Machine Learning Library und die Java Komponenten sind verantwortlich für die Kommunikation, Speicherung und das User Interface. Zur Aufzeichnung der Daten werden die drei Sensoren, GPS, Beschleunigungssensor und Mikrofon genutzt. Das Mikrofon wird verwendet um soziale Interaktionen wie sprechen und nicht sprechen zu erkennen. Zum Klassifizieren von physischen Aktivitäten wird der Beschleunigungsensor genutzt. Damit ist es möglich zu erkennen ob die Person, fährt, läuft, geht oder sich nicht bewegt. Die Daten des GPS-Sensors werden nicht zur Klassifizierung genutzt sondern nur aufgezeichnet, um die Positionen der Person ermitteln zu können. Zusätzlich existiert noch ein eigenes *Schlafmodel*, dass versucht die Schlafdauer einer Person zu modellieren. Dafür wird die Häufigkeit und Dauer von Akkuaufladungen, sowie die Zeitspannen, in denen sich das Handy nicht bewegt und sich in einer ruhigen Umgebung befindet. Diese Daten werden jedoch nicht laufend erhoben sondern nur einmal täglich. Die gesamten Daten werden in unabhängigen SQLite Dateien gespeichert. Diese Dateien werden dann an die Cloud Infrastruktur geschickt wo sie weiter verarbeitet und aufbereitet werden. Um die Akkulaufzeit des Smartphones zu schonen werden diese

Dateien nur hochgeladen, wenn der Akku sich gerade im Ladezustand und das Smartphone sich im WiFi befindet. [Lane et al., 2011]

4.5.2 Smartphone Applikation

Bei der Smartphone Applikation handelt es sich um eine vereinfachte Version der Web Applikation die in 4.5.4 beschrieben wird. Der Benutzer kann damit seine aktuell und in der Vergangenheit erreichten Punkte ansehen. Zusätzlich kann der Benutzer sich noch Trends seine Verhaltens und seiner Punkte anzeigen lassen. [Lane et al., 2011]

4.5.3 Hintergrundbild

Zusätzlich zur Smartphone Applikation existiert noch eine weniger interaktive Ansicht der Daten mithilfe eines Aquariums. Da dieses Aquarium als Hintergrundbild und Speerbildschirm verwendet wird kann man dem Benutzer ständig Feedback geben ohne die Applikation starten zu müssen. Andere Projekte wie das Ubifit Garden Projekt [Consolvo et al., 2008] haben bereits herausgefunden, dass die Visualisierung der Daten über das Hintergrundbild sehr effektiv sein können. In der Visualisierung werden Punkte für Schlafen, Aktivitäten und soziale Interaktionen vergeben. Jede dieser drei Komponenten wird durch ein anderes Lebewesen abgebildet.

Schildkröte: Die Schildkröte visualisiert die Schlafgewohnheiten des Benutzers. Wenn die Schildkröte schläft bedeutet das, dass der Benutzer zu wenig Schlaf hat. Kommt sie jedoch aus ihrem Panzer heraus und beweget sich, zeigt das, dass der Benutzer genügend Schlaf erhalten hat.

Clownfisch: Mit dem Clownfisch werden die physischen Aktivitäten des Benutzers visualisiert. Die erreichte Punktzahl beeinflusst dabei die Bewegungen und die Geschwindigkeit des Fisches. Wenn der Benutzer zu wenige Aktivitäten aufweist bewegt sich der Fisch sehr langsam. Je höher die Punktzahl umso energischer bewegt sich der Fisch. Außerdem beginnt der Fisch bei einer sehr hohen Punktzahl mit Saltos und Backflips.

Fischschwarm: So wie auch der Clownfisch bewegt sich der Fischschwarm über den Bildschirm. Dabei wächst die Anzahl der Fische mit den sozialen Interaktionen. Außerdem schwimmen der Clownfisch und der Fischschwarm näher beieinander, wenn sich die sozialen Interaktionen steigern.

4.5.4 Web Applikation

Den BeWell Benutzer ist es möglich ihre Daten, zusätzlich zur mobile Applikation, über ein Web-Portal anzusehen. Das Portal bietet hauptsächlich zwei Bereiche:

- Tagebuchähnliche Visualisierung der vom System erkannten Verhaltensmuster und Visualisierung der erreichten Punkte.
- Medizinische Standard-Fragebögen zur Kontrolle der Niedergeschlagenheit, des Schlafs und des Wohlbefindens des Benutzers.

Die tagebuchähnlichen Visualisierung erlaubt es dem Benutzer seine aufgezeichneten Daten anzusehen und diese auch noch zu bearbeiten. Der Benutzer kann dabei auf seine Positionsdaten zugreifen, um zu sehen wo er überall war, oder sich die Tonaufnahmen, die jedoch keine Konversationen beinhalten, anhören. Er kann nicht nur die aufgezeichneten Daten bearbeiten sondern auch neue Aktivitäten hinzufügen. Das ist nötig, da es nicht möglich ist Aktivitäten, wie zum Beispiel Schwimmen, automatisch zu erkennen. Dafür gibt es vorgefertigte Formulare aus denen der Benutzer irgendeine Aktivitäten des Physical Activities Compendium [Ainsworth et al., 2000] auswählen kann. Die erreichten Punkte werden dann wieder automatisch, anhand der aufgezeichneten und manuell hinzugefügten Aktivitäten, berechnet. [Lane et al., 2011]

4.5.5 Cloud Infrastruktur

Die Cloud Infrastruktur besteht aus mehreren Standard Linux Servern. Die Anbindung der Smartphones wird über ein RESTful Interface hergestellt. Dieses Interface ist dafür verantwortlich die SQLite Dateien der Smartphones entgegen zu nehmen und zu verarbeiten. Außerdem können die berechneten Daten wieder über dieses Interface abgefragt werden. [Lane et al., 2011]

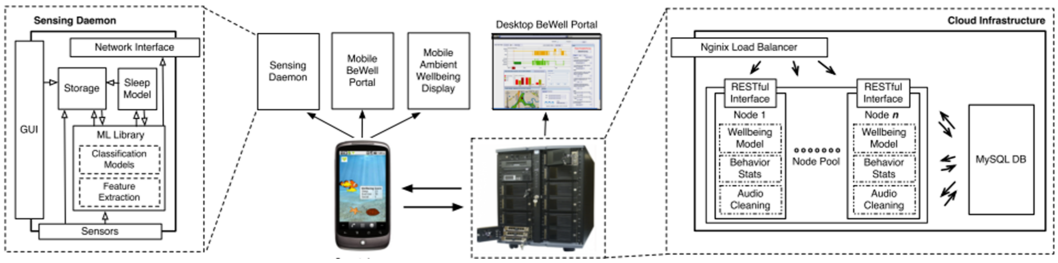


Abbildung 4.6: Architektur der BeWell Komponenten. [Lane et al., 2011]

4.6 Zusammenfassung

Wie in Tabelle 4.1 zu sehen ist gibt es unterschiedlichste Applikationen oder auch Frameworks die sich unterschiedlichster Sensoren bedienen, um die verschiedensten Fragestellungen zu beantworten. Das SoundSense Projekt ist hierbei jedoch gesondert zu behandeln. Hier wurde keine Lösung eines bestimmten Problems entwickelt sondern ein spezieller Sensor, in diesem Fall das Mikrofon, verwendet und veranschaulicht was alles mit diesem einen Sensor möglich ist. Wie in Kapitel 5 oder Kapitel 3 zu sehen ist, findet man im Projekt iPeeper oder im „SensorKit“ Framework eine ähnliche Herangehensweise. Der Unterschied liegt darin, dass nicht nur ein Sensor sondern alle bzw. möglichst viele interne Sensoren verwendet wurden.

	Jigsaw	JigMe	GreenSaw	CenceMe	UbiFit Garden	SoundSense	BeWell	iPeeper
Art des Projektes								
Applikation		x	x	x	x	x	x	x
Framework	x					x		
Betriebssystem								
iOS	x	x	x			x		x
Android							x	
Symbian	x	x	x	x	x			
Sensoren								
Beschleunigungssensor	x	x	x		x		x	x
GPS	x	x	x	x			x	x
Mikrofon	x	x	x	x		x	x	x
Kamera				x				x
Bluetooth				x				
Barometer					x			
Benutzereingaben							x	x
Externe Sensoren					x			
Persisitierung und Verarbeitung								
Persisitierung		?	x		x	x	x	x
Verarbeitung am Server				x			x	
Verarbeitung am Smartphone	x	x	x	x	x	x		x
Visualisierung								
Smartphone			x		x		x	x
Web Portal							x	
Soziales Netzwerk		x		x				

Tabelle 4.1: Übersicht aller ähnlichen Projekte

Kapitel 5

Prototyp - iPeeper

Im Zuge dieser Masterarbeit ist, über die beiden entwickelten Frameworks *SensorKit* und *SensorModel* hinaus, ein voll funktionsfähiger Prototyp namens “iPeeper” entstanden. Dieser, in diesem Kapitel vorgestellte Prototyp, ist das Ergebnis aus den Masterarbeiten von [Bachmann, 2012], der sich mit der Visualisierung von mobilen Sensor Daten befasst hat, und mir. Zusätzlich zu den entwickelten Frameworks wurde ein benutzerfreundliches User Interface für iPhone und iPad umgesetzt. Außerdem wurde noch die Möglichkeit geschaffen die aufgezeichneten Daten an ein anderes iOS Gerät zu senden und somit seine Daten zu verteilen.

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 Georg Bachmann entwickelte *VisualizationKit (VK)* sowie das von mir entwickelte *SensorKit(SK)* und *SensorModel (SM)* Framework.

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

5.1 Übersicht

Bei iPeeper handelt es sich um eine iPhone/iPad Applikation um User-Context Daten aufzuzeichnen und zu visualisieren. In Kapitel 5.2 wird genauer auf die Datenaufzeichnung eingegangen und Kapitel 5.3 behandelt die Datenvisualisierung.

Das Poster, das in Abbildung 5.1 zu sehen ist, wurde für iPeeper im Zuge der i-KNOW 2011[1] erstellt und auf der Konferenz dem Publikum präsentiert.

Abbildung 5.1: Poster für iPeeper, dass auf der i-KNOW'11 Konferenz präsentiert wurde

[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)

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.

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 5.4.

5.2 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 5.5). 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.

Wie in Abbildung 5.2 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 5.3 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.

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

Abbildung 5.2: Anlegen einer neuen Session

den, kann man mit dem tatsächlichen Aufzeichnen starten. Wie in Abbildung 5.3 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 3.3.1.5).

Für die Zukunft sind weitere passive Sensoren geplant, wie beispielsweise das Eintragen einer Notiz. Diese und andere mögliche Erweiterungen werden im Kapitel 7 Zusammenfassung näher beschrieben.

Wie man durch die Tabs am unteren Bildschirmrand in Abbildung 5.3 sehen kann, gibt es neben den passiven auch aktive Sensoren, die ihren aktuellen Wert anzeigen.

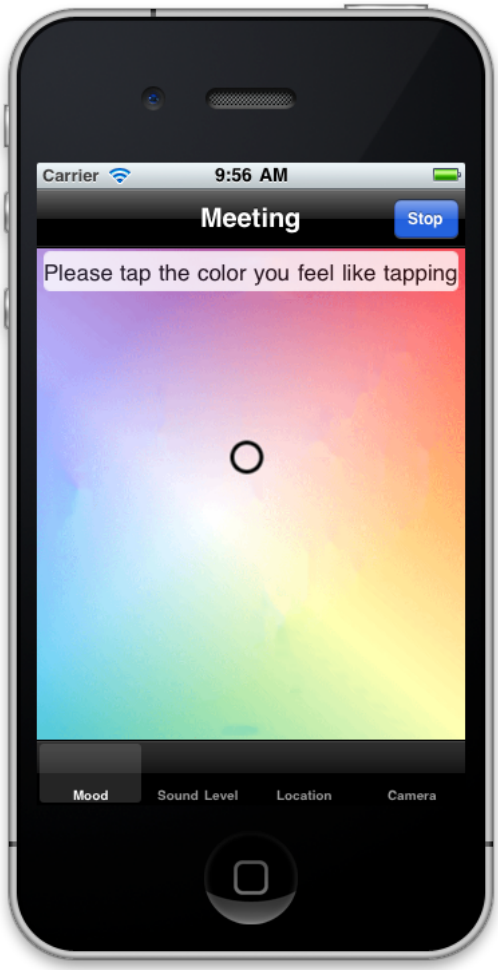


Abbildung 5.3: Aufzeichnen einer neuen Session

5.3 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 5.4 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

Abbildung 5.4: Auswahl einer Session für die Visualisierung derer Daten

den Button links unten, der in Abbildung 5.5 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 5.6 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 Reihenfolge zu machen. 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 (deren Werte Stati wie “gehen”, “stehen”, “sitzen” sein können) mit den Daten vom Bildsensor und vom GPS Modul mischt. Dazu bräuchte man eine Balken oder Torten-Diagramm Visualisierung für den Bewegungssen-

Abbildung 5.5: Auswahl der verfügbaren Visualisierungen

sor, eine Karte für die GPS Positionen und noch eine Bildvisualisierung für die Bilder.

Nachdem die Visualisierung und die dazu passenden Datensätze gewählt sind, können die Visualisierungen angezeigt werden. Das kann beispielsweise wie in Abbildung 5.7 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.

Abbildung 5.6: Auswahl der verfügbaren Daten

5.4 Datenaustausch

Beim Datenaustausch zwischen iPhone und iPad wäre es am einfachsten gewesen, das von Apple entwickelte GameKit[2] 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 im Zuge eine Doktorarbeit entwickelt wurde, auszutauschen. Des Weiteren ist eine Portierung von iPeeper auf andere mobile Plattformen, wie zum Beispiel die Android Plattform, für die Zukunft geplant und somit war es wichtig einen offenen Standart zu verwenden.

[2] http://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/GameKit_Guide

Abbildung 5.7: Anzeige der aufgezeichneten Daten in iPeeper

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[3] 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 5.4.1) zurückgreift.

Die genaue Spezifikation des Datenformates wird in Kapitel 5.4.2 beschrieben.

Abbildung 5.8: Logo des Bonjour Services

5.4.1 Finden von anderen iPeeper Instanzen

Bei der in iPeeper implementierten Lösung zum Datenaustausch kommt das soeben angesprochene Bonjour[4] zum Einsatz, um das Finden anderer Geräte in der 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 5.9). 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.

5.4.2 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 5.10 beschrieben.

Jedes Paket kann beliebig groß sein. Damit der Empfänger jedoch weiß wann er aufhö-

[3] http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices
[4] http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices

Abbildung 5.9: Versenden einer aufgezeichneten Session

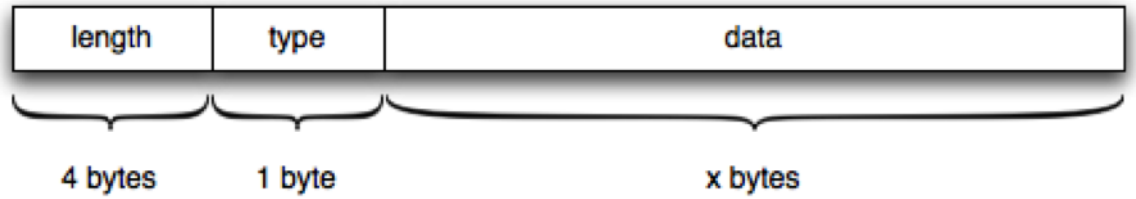


Abbildung 5.10: Aufbau von Netzwerk Paketen für den Datenaustausch in iPeeper

ren 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:

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 [5] 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 5.11 zeigt dabei den Fall, dass eine Instanz von iPeeper von sich aus Daten verschicken will, Abbildung 5.12 zeigt den gerade beschrieben Fall, in dem Daten von jemanden angefordert werden.

5.5 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 5.6), 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

[5] http://know-center.tugraz.at/ueber-uns (zuletzt besucht am 12. Juli 2012)

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

5.6 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 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.

Da die möglichen Informationen, die eine iOS Applikation auf Grund ihrer Umgebung in einer Sandbox[6], in der sie aus Sicherheitsgründen leben muss, beziehen kann, sind leider

[6] http://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide

recht limitiert. So kann iPeeper, sobald es in den Hintergrund gelegt wird, keine Daten mehr aufzeichnen, da Aktivitäten 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.

Das VisualizationKit Framework [Bachmann, 2012], dass in iPeeper Daten visualisiert, hat 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 Liniendiagramm hervorheben. Umgekehrt wäre es schön, eine passende Säule in einem Balkendiagramm hervorzuheben, wenn in einem Liniendiagramm ein Zeitpunkt zur Synchronisation ausgewählt wurde.

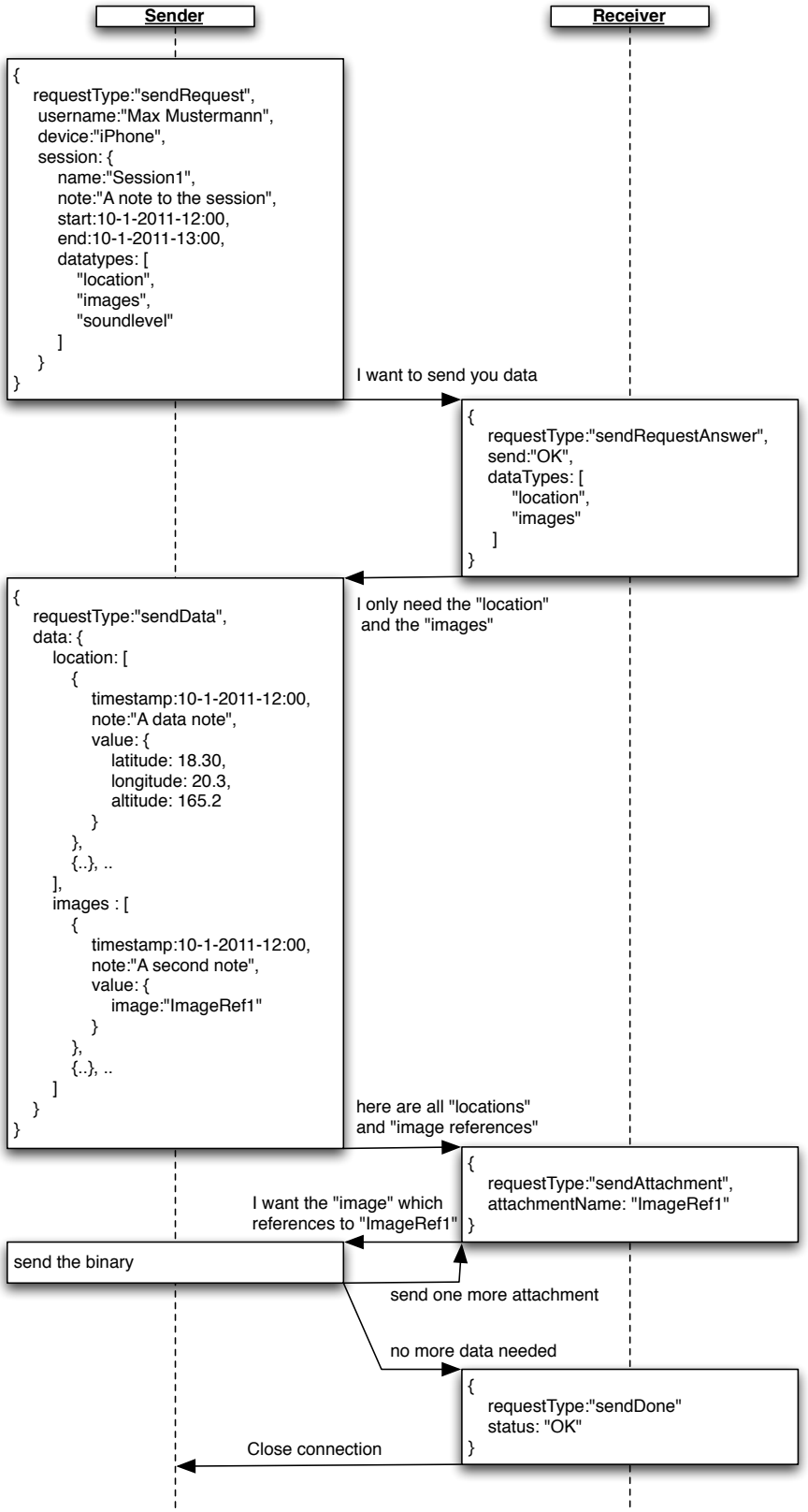


Abbildung 5.11: Netzwerk Diagramm für den Use-Case, in dem eine Instanz von iPeeper von sich aus Daten an jemanden verschicken will

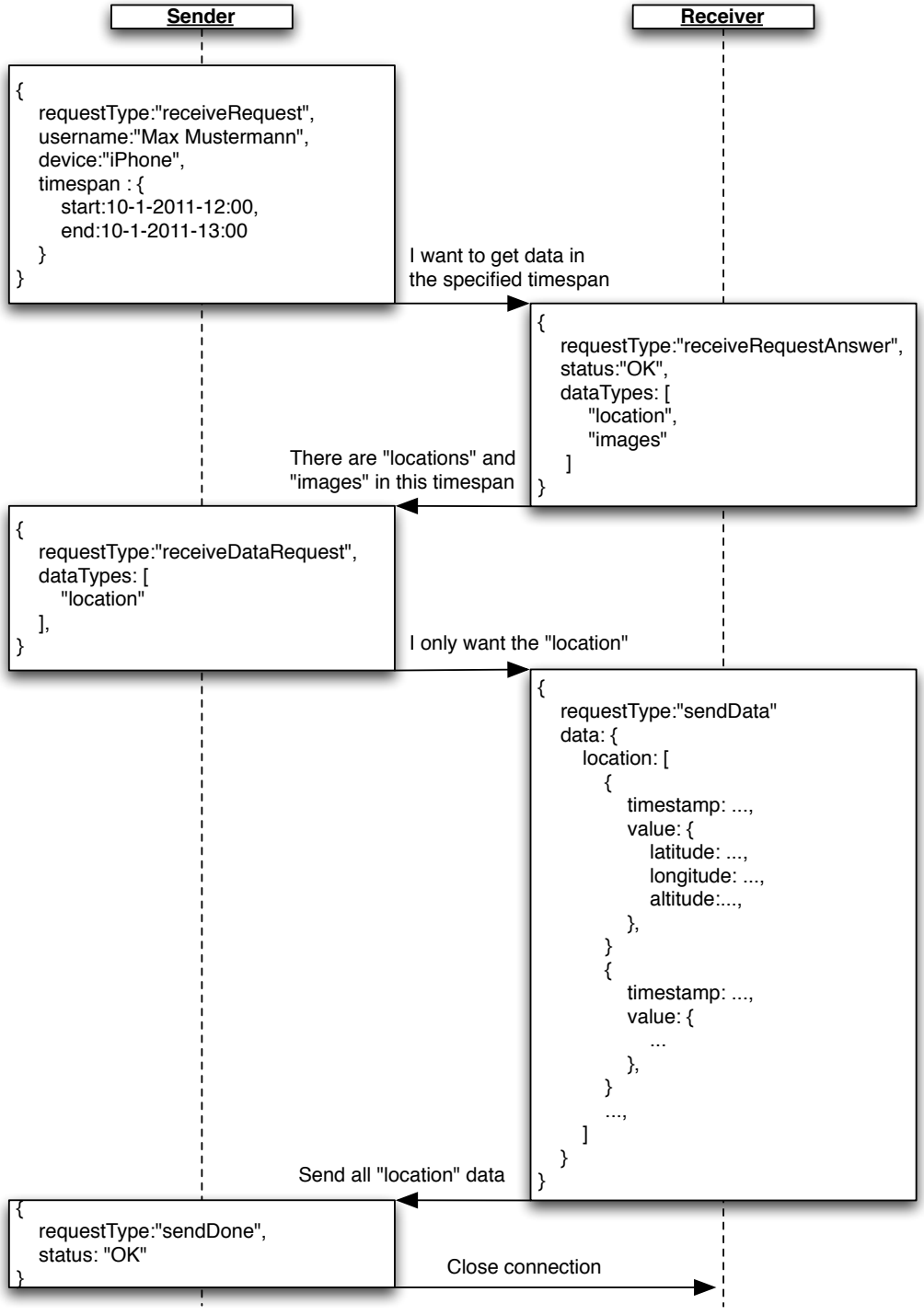


Abbildung 5.12: Netzwerk Diagramm für den Use-Case, in dem jemand Daten von einer iPeeper Instanz anfordert

Kapitel 6

Mögliche Anwendungsszenarien

Diese Anwendungsszenarien wurden gemeinsam mit Georg Bachmann [Bachmann, 2012] ausgearbeitet und kommen aus diesem Grund in dieser, wie auch in der Masterarbeit von Georg Bachmann, wortgleich vor.

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.

6.1 Zivilschutz

Dieser Use Case beschreibt wie die iPeeper App, die in Kapitel 5 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.

6.1.1 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.

6.1.2 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.

6.1.3 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.

6.1.4 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.

6.1.5 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.

6.2 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.

6.2.1 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.

6.2.2 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.

6.2.3 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.

6.2.4 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.

6.2.5 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 [Fessl et al., 2012].

6.3 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.

6.3.1 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.

6.3.2 Voraussetzungen

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

6.3.3 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.

6.3.4 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.

6.3.5 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 7

Diskussion und Ausblick

7.1 Zusammenfassung

Ziel dieser Masterarbeit war es, ein Framework zu entwickeln mit dem es dem Benutzer möglich ist mobile Sensordaten aufzuzeichnen. Außerdem sollten die aufgezeichneten Daten für die weitere Verwendung in effizienter Form gespeichert werden. Um die Daten effektiv weiter verwenden zu können sollten dafür auch diverse leicht verständlich und benutzbare Schnittstellen entwickelt werden. Obwohl versucht wurde die Schnittstellen so zu gestalten und zu benennen, dass ihre Funktion ersichtlich ist, fehlt es hier noch an Dokumentation die, den Nutzern des Frameworks, Einarbeitungszeit ersparen würde.

Eine der größten Herausforderungen dabei war die generische Architektur des Frameworks. Diese erlaubt es aber nun weitere interne, aber auch externe Sensoren in das bestehende Framework einzubinden. Dabei musste nicht nur die Architektur des Frameworks, sondern auch das Datenmodell, dass die unterschiedlichsten Senordaten speichern kann, möglichst generisch implementiert werden. Die nächste große Herausforderung war es die große Anzahl von mobilen Sensordaten möglichst effizient von den Sensoren in die Datenbank und wieder heraus zu bekommen. In all diesen Punkten mussten natürlich stets die diversen Einschränkungen der mobilen Platform akzeptiert und berücksichtigt werden.

Zu Beginn der Masterarbeit wurden die möglichen Sensoren und ihre Daten genauer untersucht, um festzustellen welche Möglichkeiten diese Daten überhaupt bieten und welche Schlüsse man damit ziehen kann. Es wurde beschlossen vorerst nur die internen Sensoren zu nutzen und lediglich die Architektur auf unterschiedliche externe Sensoren vorzubereiten. Nach diesem Entschluss wurden die wichtigsten internen Sensoren verwendet und versucht

möglichst viel Informationen aus den Daten zu gewinnen. So wurde zum Beispiel der GPS-Sensor nicht nur dazu genutzt die aktuelle Position anzuzeigen sondern auch noch um das aktuelle Wetter dieser Position abzufragen. Obwohl die Architektur theoretisch auch externe Sensoren unterstützt wurde darauf verzichtet irgendeinen zu unterstützten. Es wäre jedoch sinnvoll gewesen zumindest einen externen Sensor zu integrieren um für zukünftige Erweiterungen des Frameworks eine Referenzimplementierung bereit zu stellen.

Nachdem die ersten Sensoren ihre Daten lieferten wurde mit dem zweiten großen Teil dieser Masterarbeit begonnen. Dabei wurde das Datenbankmodell entworfen, um all diese und auch mögliche weitere Daten speichern zu können. Dabei wurden verschiedene Konzepte entwickelt, um den möglichen Performance Problemen bei Datenbankoperationen entgegen zu wirken.

Somit wurden nun bereits die ersten Daten aufgezeichnet und auch gespeichert. Dabei handelte es sich aber nur um die Rohdaten der Sensoren ohne weitere Verarbeitung. Daraufhin wurde ein neues Konzept eingeführt bei dem jeder Sensor mögliche Aggregationen implementieren kann. Das heißt es wurden weiterhin die Rohdaten in die Datenbank gespeichert, jedoch war es möglich zur Laufzeit diverse Aggregationen dieser Daten durchzuführen. Diese daraus entstandenen aggregierten Daten konnten dann, zum Beispiel für diverse Visualisierungen, genutzt werden. Dieses Konzept der Bereitstellung von aggregierten Daten ist sehr mächtig aber durch einen Fehler in der Implementierung, eines neuen Sensors, auch etwas Fehleranfällig. Aus diesem Grund sollte hier zumindest eine entsprechende Dokumentation zur Verfügung gestellt werden.

Zu guter Letzt wurde ein Prototyp namens iPeeper entwickelt der die Möglichkeiten des Frameworks zeigen soll. Dieser Prototyp wurde gemeinsam mit Georg Bachmann [Bachmann, 2012] entwickelt, dessen Masterarbeit sich mit der Visualisierung von mobilen Sensordaten beschäftigt. Da sich beide Masterarbeiten perfekt ergänzten machte es Sinn gemeinsam diesen Prototypen zu entwickeln.

7.2 Ergebnisse

Als Ergebnis dieser Masterarbeit entstand ein Framework zur effizienten Aufzeichnung und Speicherung von mobilen Sensordaten. Weiters wurde beim Entwurf der Architektur die schnelle technologische Entwicklung berücksichtigt. Das heißt die Architektur wurde möglichst flexibel und generisch gehalten, um auch zukünftige Sensoren einbinden zu können. Dadurch wird es möglich mit nur einem Framework viele Sensoren und somit viele Sens-

ordaten abzufragen und diese Daten auf effiziente Weise zu verarbeiten und zu speichern.

7.3 Ausblick

Das in dieser Masterarbeit entstandene Framework könnte als Basis für viele weitere Entwicklungen dienen. Am Wichtigsten wäre die Weiterentwicklung im Bereich der Sensoren. Hier könnten noch unzählige weitere Sensoren, vor allem die noch nicht berücksichtigten externen Sensoren, eingebunden werden um, das Framework noch interessanter zu machen. Zusätzlich könnten die bereits bestehenden Sensoren optimiert oder erweitert werden. Eine interessante Entwicklung in diesem Bereich wäre die Erweiterung des Kamera Sensors. Man sagt nicht umsonst „Ein Bild sagt mehr als tausend Worte“. So könnte man den Kamera Sensor dahin gehen erweitern, Informationen, wie Gesichter und somit Personen oder Orte, aus Bildern zu extrahieren.

Eine sehr nützliche Weiterentwicklung wäre die Aggregation der Rohdaten zu verbessern oder neue Aggregation zu implementieren. In der aktuellen Version wird vor allem die Möglichkeit gezeigt aber bis auf die Berechnung von Cluster oder Trends sind die Aggregationen noch nicht sehr weit entwickelt.

Eine andere Möglichkeit wäre ein in Kapitel 6 vorgestelltes Anwendungsszenario umzusetzen. Sehr interessant wäre hierbei das Zivilschutz Szenario da in diesem Fall die Sensoren sehr wertvolle Informationen für den Einsatz liefern können. Dafür würden die bis jetzt umgesetzten Sensoren bereits ausreichen. Man müsste hauptsächlich die Möglichkeit einer Echtzeit-Datenübertragung integrieren und die Serverkomponente entwickeln, die die Daten entgegennehmen und für die Einsatzzentrale aufbereiten kann.

Literaturverzeichnis

Ainsworth, B. E., W. L. Haskell, M. C. Whitt, M. L. Irwin, A. M. Swartz, S. J. Strath, W. L. O'Brien, D. R. Bassett, und P. O. Schmitz [2000]. *Compendium of Physical Activities: An Update of Activity Codes and MET Intensitiess*. *Medicine and Science in Sports and Exercise*, 32, Seite 2000. (Cited on page 46.)

Avanade [2011]. *Global Survey : Has Cloud Computing Matured?* Technischer Bericht. (Cited on pages v, 8 and 9.)

Bachmann, Georg [2012]. *Visualisierung von mobilen Sensordaten*. (Cited on pages 3, 49, 62, 65 and 73.)

Bell, Gordon und Jim Gemmel [2007]. *A Digial Life*. *Scientific American*, March(58 - 65). (Cited on pages v, 17 and 18.)

Bollen, Johan, Huina Mao, und Xiao-Jun Zeng [2011]. *Twitter mood predicts the stock market*. *Journal of Computational Science*. (Cited on page 33.)

Boud, David [1985]. *Reflection : Turning Experience into Learning*. Routledge Chapman & Hall, 170 Seiten. (Cited on page 61.)

Bush, Vannevar [1945]. *As We May Think*. *The Atlantic Monthly*, 176(1), Seiten 101 – 108. (Cited on page 17.)

Canalys [2012]. *Smart phones overtake client PCs in 2011*. Technischer Bericht February. (Cited on pages v, 4 and 5.)

Cherry, Steve [2005]. *Total Recall*. *IEEE Spectrum*, November. (Cited on pages v and 17.)

Choudhury, Tanzeem, Jeffrey Hightower, Anthony Lamarca, Louis Legrand, Ali Rahimi, Adam Rea, Bruce Hemingway, Karl Koscher, James A Landay, Jonathan Lester, und Danny Wyatt [2008]. *The Mobile Sensing Platform: An Embedded System for Capturing and Recognizing Human Activities*. Seiten 32–41. (Cited on pages 5 and 41.)

Consolvo, Sunny, David W Mcdonald, Tammy Toscos, Mike Y Chen, Jon Froehlich, Beverly Harrison, Predrag Klasnja, Anthony Lamarca, Louis Legrand, Ryan Libby, Ian Smith, und James A Landay [2008]. *Activity Sensing in the Wild : A Field Trial of UbiFit Garden*. In *Proc. of the 26th SIGCHI Conference on Human Factors in Computing Systems*. ISBN 9781605580111. (Cited on pages vi, 41, 42 and 45.)

Cuervo, Eduardo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Chandra Ranveer, und Bahl Paramvir [2012]. *MAUI : Making Smartphones Last Longer with Code Offload*. In *Proc. 8th ACM MobiSys*. ISBN 9781605589855. (Cited on page 11.)

Cuff, By Dana, Mark Hansen, und Jerry Kang [2008]. *Urban Sensing : Out of the Woods*. *Communications of the ACM*, 51(3), Seiten 24 – 33. (Cited on page 5.)

Eagle, Nathan [2008]. *Behavioral inference across cultures: using telephones as a cultural lens*. *IEEE Intelligent Systems*, 23, Seiten 62–24. (Cited on page 5.)

Economist [2012]. *The quantified self - Counting every moment*. (Cited on page 16.)

Eisenman, Shane B, Nicholas D Lane, Emiliano Miluzzo, Ronald A Peterson, Gahng-seop Ahn, und Andrew T Campbell [2006]. *MetroSense Project : People-Centric Sensing at Scale*. In *Proceedings of the ACM Sensys World Sensor Web Workshop*. ISBN 1595933433. (Cited on page 5.)

Fessl, A., V. Rivera-Pelayo, V. Pammer, und S. Braun [2012]. *Mood Tracking in Virtual Meetings*. In *7th European Conference on Technology Enhanced Learning - EC-TEL 2012*. (Cited on pages 30 and 70.)

Fogg, B. J. [2002]. *Persuasive Technology: Using Computers to Change What We Think and Do*. 1 Auflage. Morgan Kaufmann, 312 Seiten. (Cited on page 15.)

Gemmell, Jim, Gordon Bell, Roger Lueder, Stefan Drucker, und Curtis Wong [2002]. *MyLifeBits: fulfilling the Memex vision*. In *Proceedings of the tenth ACM international conference on Multimedia*, Seiten 235 – 238. (Cited on page 17.)

Hammer-Lahav, E. [2010]. *The OAuth 1.0 Protocol*. (Cited on page 33.)

Koekkoek, Hendrik [2011]. *Distimo Publication Full Year 2011*. Technischer Bericht, Distimo. (Cited on pages v and 6.)

Lane, Nicholas D, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, Andrew T Campbell, und Dartmouth College [2010]. *A Survey of Mobile Phone Sensing*. *IEEE Communications Magazine*, (September), Seiten 140–150. (Cited on pages v, 4, 5, 9, 10 and 13.)

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*. In *5th International ICST Conference on Pervasive Computing Technologies for Healthcare*. (Cited on pages v, vi, 14, 44, 45 and 46.)

Lu, Hong, Wei Pan, N.D. Lane, Tanzeem Choudhury, und A.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. ACM. (Cited on pages vi and 43.)

Lu, Hong, Jun Yang, Zhigang Liu, N.D. Lane, Tanzeem Choudhury, und A.T. Campbell [2010]. *The Jigsaw continuous sensing engine for mobile phone applications*. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*, Seiten 71–84. ACM. (Cited on pages 38 and 39.)

Miluzzo, Emiliano, N.D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, S.B. Eisenman, Xiao Zheng, und A.T. Campbell [2008]. *Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application*. In *Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys '08)*, Seiten 337–350. ACM, Raleigh, NC, USA. (Cited on pages vi, 11, 39 and 40.)

Miluzzo, Emiliano, N.D. Lane, Hong Lu, und A.T. Campbell [2010]. *Research in the App Store Era: Experiences from the CenceMe App Deployment on the iPhone*. In *Proc. of The First International Workshop Research in the Large: Using App Stores, Markets, and other wide distribution channels in UbiComp research*. Citeseer. (Cited on page 5.)

Miluzzo, Emiliano, Nicholas D Lane, Shane B Eisenman, und Andrew T Campbell [2007]. *CenceMe – Injecting Sensing Presence into Social Networking Applications*. In *Second European Conference on Smart Sensing and Context*, Seite 28. (Cited on page 40.)

Priyantha, Bodhi, Dimitrios Lymberopoulos, und Jie Liu [2010]. *Little Rock : Enabling*

Energy Efficient Continuous Sensing on Mobile Phones. Technischer Bericht, Microsoft Research, Redmond, WA. (Cited on pages v and 12.)

Rath, Andreas [2010]. *User Interaction Context*. Dissertation, University of Technology Graz. (Cited on page 56.)

Russel, James A. [1980]. *A circumplex model of affect. Journal of Personality and Social Psychology*, Seiten 1161–1178. (Cited on pages vi and 30.)

Sadun, Erica [2012]. *The iOS 5 Developer's Cookbook*. Addison-Wesley, 790 Seiten. (Cited on pages v, 20, 21, 22, 23 and 24.)

Shilton, Katie [2009]. *Four billion little brothers? Communications of the ACM*, 52(11), Seite 48. ISSN 00010782. (Cited on page 5.)

Stahl, Anna, Petra Sundström, und Höök Kristina [2005]. *A Foundation for Emotional Expressivity*. In *Proceedings of the 2005 conference on Designing for User eXperience*. AIGA: American Institute of Graphic Arts. (Cited on page 30.)