Ausarbeitung und Umsetzung einer Experimentalumgebung

zur Benutzeranalyse in mobilen Umgebungen

Diplomarbeit im Fach Informatik

vorgelegt von

Mathias Graf
Wollerau, Schweiz
Matrikelnummer: 99-727-018

Angefertigt am

Institut für Informatik der Universität Zürich
Prof. Dr. A. Bernstein

Betreuer: Peter Vorburger

Abgabe der Arbeit: 3. Mai 2006

Zusammenfassung

In unserem täglichen Leben haben mobile Kommunikationsgeräte einen sehr hohen und noch immer wachsenden Stellenwert eingenommen. Der grosse Vorteil, dass man nun immer und überall jemanden erreichen kann und wir so viel flexibler leben können, hat aber auch einen hohen Preis: All zu oft werden wir in den ungünstigsten Situationen von belanglosen Anrufen gestört und in unserer Tätigkeit unterbrochen.
Diese Diplomarbeit hat zum Ziel die Rahmenbedingungen für ein Langzeitexperiment festzulegen mit welchem Daten über das Benutzerverhalten gesammelt werden. Anhand diesen Daten kann überprüft werden, ob es möglich ist, die Unterbrechbarkeit eines Benutzers vorherzusagen.
Diese Rahmenbedingungen werden einerseits durch die dem Experiment zugrunde liegende Methodik und andererseits durch die sich aus der Implementierung entstehenden Möglichkeiten und Einschränkungen festgelegt.
Um die Machbarkeit zu überprüfen, musste für die Umsetzung des Experimentaufbaus eine Applikation erschaffen werden, welche die für das Experiment benötigten Anforderungen erfüllt.

Abstract

In our daily lifes, mobile communication devices attain very high and still growing significance. This gives us the ability to reach anybody, everywhere at anytime. Consequently, also insignificant calls reach and disturb us in unfavorable situations.
The goal of this Thesis is to determine the key parameters for a long-term experiment to gather data on user behavior. The analysis of these data may help to predict the interruptability level of a user and, subsequently, taking appropriate measures.
These key parameters are determined on the one hand by the experimental method and on the other hand by the features of the implementation environment.
An application has been created in order to assess the feasibility of the experiment setup.

Inhaltsverzeichnis

1 Einleitung

Die rasante Entwicklung im Bereich der mobilen Kommunikation, hat dazu geführt, dass Kommunikationsgeräte wie Handys oder Smartphons sich als einen festen Bestandteil unserer Gesellschaft etabliert haben. Diese mobilen Kommunikationsgeräte sind kaum mehr aus dem heutigen Leben wegzudenken und erleichtern uns viele alltägliche Situationen. Nebst den vielen Vorteilen bringt diese neue Kommunikationsform aber auch einige nicht zu unterschätzende Nachteile mit sich. So bringt die Möglichkeit, dass man immer und überall erreicht werden kann, auch das Risiko mit sich, dass man in beliebigen Situationen unerwünscht gestört und in seiner Tätigkeit unterbrochen werden kann. So wird z.B. ein Anruf während dem Essen in einem guten Restaurant, unabhängig von der inhaltlichen Wichtigkeit des Gesprächs, als Belästigung empfunden.

Folgendes Szenario kann sich täglich irgendwo abspielen und illustriert die Problematik noch genauer:

> *Ein Manager des mittleren Kaders, in einer streng hierarchisch gegliederten Firma, ist in einer wichtigen Sitzung, in welcher er der Unternehmensleitung den Stand der Projekte, für welche er die Verantwortung trägt, präsentieren muss. Je nach Ausgang dieser Präsentation werden weitere finanzielle Mittel für die Projekt genehmigt oder auch nicht. Der Manager ist nun aber in der misslichen Lage, dass er dringend noch auf Informationen eines Projektleiters wartet, welcher ihm die neusten Ergebnisse eines zentralen Projektes noch nicht mitgeteilt hat.*
> *Er muss sich nun entscheiden, ob er während der Sitzung entweder sein Handy auf lautlos stellt, so aber Gefahr läuft, dass er die essentiellen Informationen nicht mehr rechtzeitig erhält,. oder ob er den Ton des Handys aktiviert lässt, was aber zur Folge haben kann, dass er von einem belanglosen Anruf unterbrochen wird, was in dieser Situation sicher nicht angebracht wäre und er so bei der Geschäftsleitung einen schlechten Eindruck hinterlassen könnte. Egal wie er sich entscheidet, die Situation ist in beiden Fällen nicht optimal.*

Dieses Beispiel zeigt, dass in dieser und auch in vielen anderen weniger prägnanten Situationen ein Telefonfilter, welcher ungewollte oder störende Anrufe blockiert und erwünschte durchlässt, sehr wünschenswert und hilfreich wäre.
So wäre es denkbar, dass in nicht allzu ferner Zukunft auf dem Handy im Hintergrund ein Programm läuft, welches, um zu unserem Beispiel zurückzukehren, dem Manager nur den Anruf des Projektleiters durchstellt und alle anderen Anrufe direkt auf die „Combox“ umleitet.

1.1 Context Awareness / Unterbrechbarkeit

Um einen solchen Filter entwickeln zu können, muss der Kontext, in welchem sich die Person gerade befindet, erkannt werden, um so anhand dieser Informationen entscheiden zu können, wie der Filter handeln soll. Da vor dem Anruf der Inhalt des Gesprächs nicht bekannt ist, kann ein Inhaltsfilter, wie bei Email-Filter, nicht angewandt werden. Deshalb ist für diesen Filter der Kontext umso wichtiger.
Um den Begriff Kontext genauer einzugrenzen folgt ein Zitat von Anind K. Dey in welchem er kurz und prägnant den Begriff Kontext definiert:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.* [Dey]

Sinngemäss auf Deutsch übersetzt bedeutet das, Kontext irgendeine Information ist, welche dazu benutzt werden kann, die Situation eines Objektes zu charakterisieren. Ein solches Objekt kann eine Person, ein Ort oder ein Gegenstand sein, welcher als relevant für die Interaktion zwischen dem Benutzer und einer Applikation ist. Auch der Benutzer und die Applikation selbst kann als ein solches Objekt betrachtet werden.

Da aus der Sicht des Filterprogramms der Kontext der betreffenden Person nicht bestimmt werden kann, wird der Kontext des Kommunikationsgerätes betrachtet. Der Zustand in welchem sich das Handy gerade befindet kann mittels Informationen, welche über das Gerät durch interne und externe Sensoren gewonnen wurden, klassifiziert werden. Durch den nun bekannten Kontext des Telefons können nun gewisse Rückschlüsse auf den Kontext, in welchem sich die Person gerade befindet, getroffen werden.

Um ein solches Filterprogramm realisieren zu können, muss nun, anhand des ermittelten Kontextes, die Unterbrechbarkeit bestimmt werden können. Die Unterbrechbarkeit ist ein innerer Zustand einer Person oder auch eines Objektes. Dieser Zustand bestimmt, ob und unter welchen Voraussetzungen die Person gestört werden darf.

1.2 Vision

Die Vision ist es nun, ein solches Programm zu entwickeln, welches selbständig diese Filterfunktion für uns übernimmt und selbständig entscheiden kann, ob wir im Moment durch den eingehenden Anruf unterbrochen werden können oder nicht.

Diese Applikation, könnte etwa nach folgendem Schema aufgebaut sein und folgende Funktionalitäten erfüllen:

(Abb. 1: Ablauf der zukünftigen Applikation)

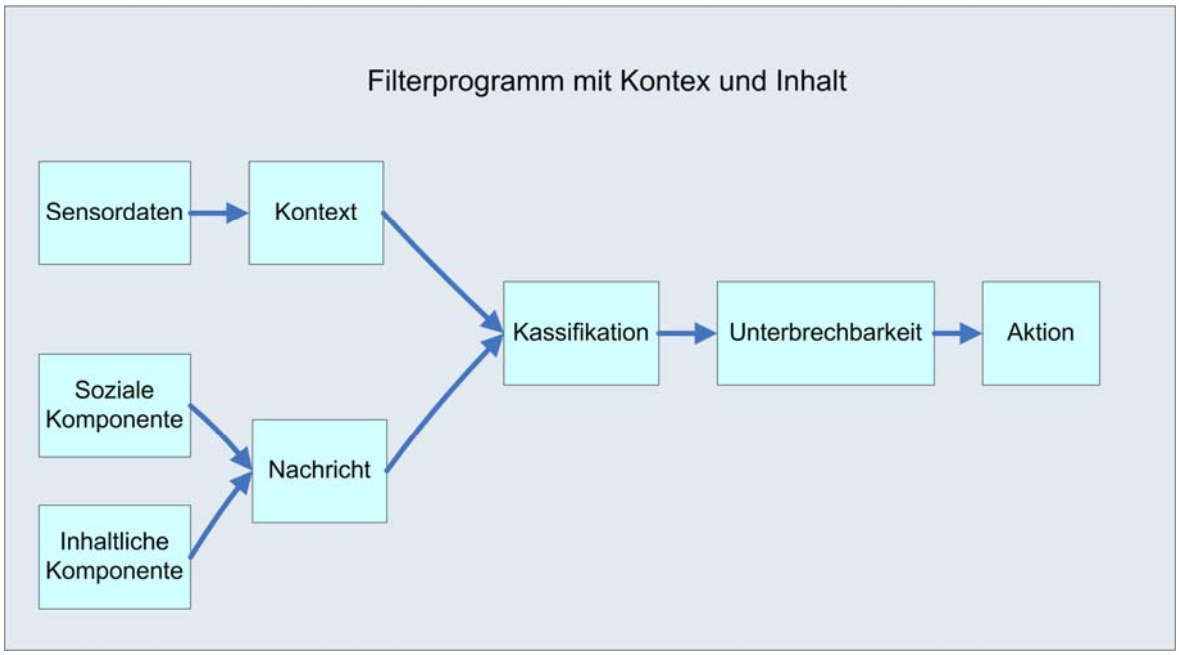
Im Hintergrund werden laufend Sensoren abgefragt und mittels dieser Daten eine Klassifizierung vorgenommen um den Kontext und somit auch die Unterbrechbarkeit zu bestimmen.
Sobald nun ein Anruf eingeht muss das Programm, anhand der zuvor erstellten Klassifizierung, bestimmen, welche Aktion ausgeführt werden soll. So muss es sich entscheiden, ob es normal klingelt, nur vibriert oder den Anrufer direkt auf die „Combox“ weiterleitet soll.
Eventuell kann noch ein Belohnungs- und Bestrafungssystem eingebaut werden, mit welchem der Benutzer die Aktionen des Telefons im Nachhinein bewerten kann, um es so besser an seine Präferenzen anzupassen.

Für unser Beispiel könnte das nun folgendermassen ablaufen.

Der Manager muss sich keine Gedanken mehr darüber machen, ob er nun das Telefon lautlos stellen soll oder nicht und kann sich auf seine eigentliche Aufgabe fixieren. Sobald die Sitzung beginnt, registriert das Filterprogramm, da es in regelmässigen Abständen die momentane Uhrzeit mit dem Terminkalender vergleicht, dass ein wichtiger Event eingetragen ist. Über die Audioaufzeichnung lässt sich erkennen, dass mehre Personen im Raum sind und sprechen. Dies wird auch durch zusätzlich angebrachte Gassensoren bestätigt, da der CO-Gehalt höher ist, als wenn nur einen Person im Raum wäre. Die zuletzt empfangene GPS-Position, bevor die Verbindung zu den Satelliten unterbrochen wurde, bestätigt, dass er sich in dem Gebäude befindet, in welchem er arbeitet. Anhand von diesen und möglicherweise noch mehr

Sensorwerten, kann das Filterprogramm nun erkennen, dass er nur von wichtigen und geschäftsbezogenen Personen unterbrochen werden darf. Geht nun ein Anruf ein, erkennt das Programm anhand der unter der entsprechenden Kategorie gespeicherten Nummer, ob es sich um ein geschäftliches oder ein privates Gespräch handelt. So würde ein Anruf von einer unbekannten Nummer, z.B. von Meinungsforschungsumfragen, oder sogar eines privaten Kontaktes, wie der Anruf seiner Freundin, nun direkt auf die „Combox" umgeleitet werden. Der Anruf des Projektleiters wird aber, anhand der unter der Kategorie „geschäftlich" gespeicherten Nummer, als wichtig eingestuft und durchgestellt. So erhält der Manager nun seine wichtige Information, ohne Gefahr zu laufen, durch einen unwichtigen Anruf während seiner Präsentation gestört zu werden.

Es ist sogar denkbar, dass diese Programmidee noch weiter entwickelt wird, so dass ein Programm das Telefon selbst abnimmt und der Anrufer erst dem Programm sagen muss, wer er (soziale Komponente) und was sein Anliegen (inhaltliche Komponente) ist. Die Applikation könnte nun, zusätzlich zu der Klassifikation über die Sensorwerte, mittels Spracherkennung und Stichwortfilter versuchen, die Wichtigkeit des Anrufes einzustufen und dem entsprechend zu handeln. Der Ablauf eines solchen Filterprogramms, welches sowohl den Kontext als auch den Inhalt beachtet könnte wie in dem folgenden Diagramm dargestellt ablaufen.



(Abb. 2: Filterprogramm mit Kontext und Inhalt)

1.3 Einführung: Context Awareness Experiment

Um einen weitern Schritt in Richtung dieser Vision zu tätigen, muss gezeigt werden, dass anhand von Sensordaten der Kontext des Handys klassifiziert und daraus die Unterbrechbarkeit einer Person ermittelt werden kann.
Für diesen Zweck ist ein Experiment mit einem Smartphone geplant, in welchem vorerst nur Benutzerdaten gesammelt werden. Diese Daten bestehen einerseits aus Sensordaten[1], die fortlaufend aufgenommen werden und andererseits aus Angaben des Benutzers selbst. Sobald ein Anruf eingeht beantwortet der Proband anschliessend in einem Menu Fragen zu seiner Unterbrechbarkeit.
Anhand dieser Daten gilt es, nach dem Experiment zu zeigen, dass von den Sensorwerten auf den Kontext des Benutzers und auf dessen Unterbrechbarkeit geschlossen werden kann.

Ein sehr ähnliches Experiment ist bereits durchgeführt worden, in welchem anstatt eines mobilen Kommunikationsgerätes, ein fest angeschlossenes Telefon in einem mit Sensor versehenen Raum betrachtet wurde. [Peter Vorburger, 2005]

1.4 Zielsetzung der Diplomarbeit

Durch die Vorarbeit von früheren Diplomanden[2] sind viele einzelne Elemente hin zu diesem Experiment entwickelt worden. Im Rahmen dieser Diplomarbeit sollen die einzelnen Bausteine zu einem Ganzen zusammengefügt und so ergänzt werden, dass das Experiment durchgeführt werden kann. Dies beinhaltet, dass die Machbarkeit überprüft und die Rahmenbedingung für dieses Experiment festgelegt wird.
Um die Machbarkeit überprüfen zu können musste für die Umsetzung des Experimentaufbaus eine Applikation erschaffen werden, welche die für das Experiment benötigten Anforderungen erfüllt.
Die Rahmenbedingungen werden nun einerseits durch Designentscheidungen und andererseits durch, die aus der Implementierung entstehenden, Möglichkeiten und Einschränkungen bestimmt.

[1] Eine Auflistung und Erläuterung aller verwendeten Sensoren findet sich im Kapitel 2.3
[2] Die Wichtigsten namentlich genannt sind: Urban Kägi, Joachim Fornallaz, Frederic de Simoni, Manuel Donner und Robin Bucciarelli

1.5 Aufbau der Diplomarbeit

Nach der Einleitung wird im **Kapitel 2** die für das Experiment verwendete Hardware vorgestellt und die Auswahl erläutert. Im Speziellen werden alle Sensoren, welche zur Aufnahme während des Experimentes verwendet werden, aufgelistet und genauer erklärt.

Im **Kapitel 3** wird der Applikationsaufbau theoretisch erklärt und die einzelnen für diese Applikation benötigten Module aufgezeigt und erläutert. Zusätzlich wird eine Einführung in die Palm-Programmierung gegeben, um den Erläuterungen des nächsten Kapitels folgen zu können, mit Schwergewicht auf die Funktionen welche für die Umsetzung der Applikation eine zentrale Rolle spielten.

Im **Kapitel 4** folgt die Implementierung der einzelnen Module. Hier werden die Funktionsweise, das Vorgehen, die aufgetretnen Probleme und deren Lösungen aufgezeigt

Darauf folgt das **Kapitel 5**, in welchem die durch die Implementierung und durch die Designentscheidungen sich ergebenden Rahmenbedingungen festgehalten werden.

Kapitel 6 beinhaltet eine Zusammenfassung, die Limitierungen des Programms und ein Ausblick.

2 Aufbau des Experimentes

2.1 Wahl des Kommunikations-Gerätes

Zur Wahl des mobilen Kommunikations-Gerätes wurde von Frederic de Simoni in seiner Diplomarbeit eine Evaluation durchgeführt. Die Entscheidungskriterien waren die verfügbaren API-Dokumentation, die Implementierung der seriellen Schnittstelle und die Akku-Laufzeit des Gerätes.
Da Treo 650 das einzige der evaluierten Smartphones ist, bei welchem im durchgeführten Test erfolgreich Sensordaten über die serielle Schnittstelle eingelesen werden konnten, fiel die Wahl schlussendlich auf den Treo 650.
Gegen eine Implementierung unter PalmOS steht eine verhältnismässig aufwendige C-Programmierung (im Vergleich zu SPV-Geräten, welche in C# programmiert werden können) und das Fehlen von Multi-Threading [Simoni, 2005]

2.2 Treo 650

Der Treo 650 ist ein Smartphone, welches auf dem Palm Betriebssystem basiert. Neben den genauen Spezifikationen, welche sich im Anhang befinden, ist für unseren Zweck zu erwähnen, dass der Treo 650 ausgestattet ist mit einer VGA-Digitalkamera (max Auflösung 640 x 480), einem Mikrophon und einer sehr flexiblen Multiconnectorschnittstelle, über welche sich ein serielles Gerät anschliessen lässt. Er verfügt über einen 32 MB grossen internen Speicher und zusätzlich über einen SD-Karten Slot, in welchem Karten bis zu einer Grösse von 2 GB erkannt werden.
[Palm, 2006]

(Abb. 3: Treo 650)

2.3 Sensoren

Sobald das Experiment durchgeführt ist, lassen sich die wichtigen und aussagekräftigsten Sensoren ermitteln. Vielleicht genügen einige wenige Sensoren um den Zustand des Telefons ermitteln zu können. Aber vor dem Experiment wissen wir nicht, welche der Sensoren am aussagekräftigsten sein werden, daher brauchen wir so viele Sensoren wie möglich, um den momentanen Zustand des Telefons möglichst genau klassifizieren zu können.
Wie bereits erwähnt, unterteilen sich die Sensoren in zwei Gruppen. Die einen sind bereits im Treo integriert (interne Sensoren), die anderen werden zusätzlich hinzugefügt (externe Sensoren).

2.3.1 Interne Sensoren

Bereits im Telefon vorhanden ist ein Mikrophon. Mit diesem lässt sich die akustische Umgebung des Benutzers aufnehmen und später auch analysieren. So lässt sich ermitteln, ob z.B. mehrere Personen in seiner Nähe waren, ob gerade eine Diskussion stattgefunden hat einschliesslich evtl. sogar der Thematik des Gesprächs, sofern eine gute Spracherkennungssoftware eingesetzt wird.
Der Treo 650 bietet als weiteren Sensor eine integrierte VGA Camera. Mit dieser lässt sich die optische Umgebung des Telefons ermitteln. So können z.B. anhand der Umgebungshelligkeit, evtl. in Kombination mit anderen Sensoren, Rückschlüsse auf Ort oder Tageszeit gezogen werden oder durch Vergleich von zeitlich verschieden Bildern können Bewegungsunterschiede erkannt werden.[Simoni, 2005]
Zusätzlich können einige internen Variabeln des Treo 650 ausgelesen werden. Erfasst werden: Die Signalstärke, der Batteriestatus, ob die Tastensperre aktiv war und ob der Ton des Telefons aktiviert war. Diese Werte lassen auch einige Rückschlüsse zu. Ist der Batteriestand sehr niedrig, wird der Benutzer z.B. unwichtige Anrufe nicht entgegennehmen, um die verbleibende Energie für wichtige Telefonate einsetzen zu können. Wird das Telefon vom Benutzer bewusst auf „Mute“ gestellt, kann dies ebenfalls ein Zeichen sein, dass er nun nur noch von wichtigen Anrufen gestört werden möchte. Anhand der Tastensperre lässt sich erkennen, ob kurz vor dem Anruf noch etwas anderes mit dem Telefon gemacht wurde, wie z.B. ein Termin erfasst.
Ebenfalls ein sehr interessanter Anhaltspunkt, in welchem Zustand sich der Benutzer im Moment des Anrufes befindet, ist die Zeit, wann der Anruf stattfand, da mitten in der Nacht oder während dem Mittagessen ein Anruf sicher störend wäre. Aber auch der Vergleich der Zeit des Anrufes mit dem Terminkalender des Telefons kann Aufschluss über den Grad der Störung geben.
Die CellID wäre auch ein interessanter Sensor, welcher über den Aufenthaltsort und die Bewegungen des Benutzers des Telefons Auskunft geben würden, doch lässt sich gemäss der Diplomarbeit von Frederic de Simoni [Simoni, 2005] die CellID, hier in der Schweiz, nicht auslesen.
Der Übersicht halber werden in der folgenden Aufzählung nochmals alle internen Sensoren aufgelistet:

- Audio
- Photo
- Interne Variabeln
    - Signalstärke
    - Batteriestatus
    - Status: Tastensperre
    - Ton: aktiviert/deaktiviert
    - Uhrzeit
    - Kalender

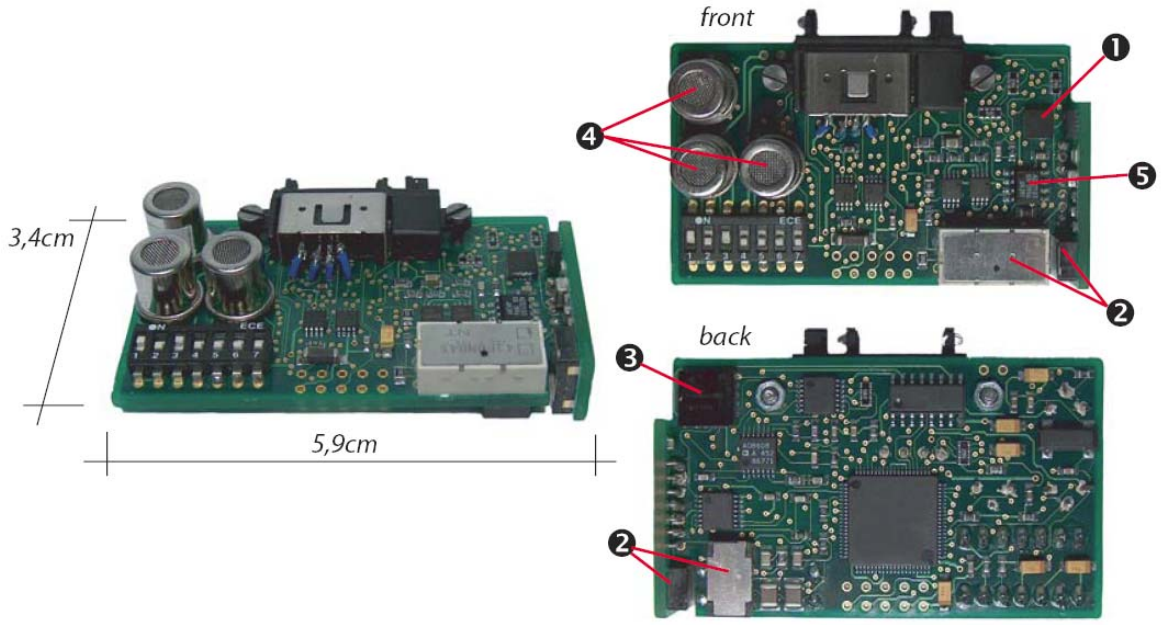
2.3.2 Externe Sensoren

Zusätzlich zu diesen internen Sensoren wird über die Bluetooth-Schnittstelle ein GPS Empfänger gekoppelt. So lassen sich Aussagen über den Ort und die Bewegung des Telefonbenutzers machen. Legt er in kurzer Zeit eine grosse Strecke zurück, lässt sich z.B. darauf schliessen, dass er momentan mit dem Auto auf der Autobahn unterwegs ist.

Um das Telefon noch mit weiteren Sensoren ausstatten zu können, wurde von Peter Vorburger eigens ein Sensorboard entwickelt, welches sich über die serielle Schnittstelle des Treo's anschliessen lässt.

(Abb. 4: Treo 650 mit über der seriellen Schnittstelle angeschlossenem Sensorboard)

Dieses Board ist mit zwölf Sensoren und fünf verschieden Sensorarten bestückt. Auf diesem Board sind drei Accelerometer [1] verbaut um eine beliebige 3D-Beschleunigung erfassen zu können. Um die Rotation des Telefons in allen drei Achsen erfassen zu können, sind drei Gyrosensoren [2] installiert. Zusammen mit den drei Magnetfeldsensoren [3] lassen sich die Richtung der Beschleunigung und Drehung im Vergleich zur Erdoberfläche bestimmen. So kann z.B. ermittelt werden, ob der Benutzer des Telefons gerade läuft, rennt oder still sitzt. Dies kann sogar soweit ausgebaut werden, dass mit der Hilfe dieses Sensorboardes eine Indoornavigation möglich wird. Dies wird in der Diplomarbeit von Adrian Bachmann untersucht[Bachmann, 2006].
Mittels der beiden Gassensoren[4] lässt sich der Kohlenmonoxid- (CO) und der Methan- (CH_4) Gehalt bestimmen. Durch den CO-Gehalt lässt sich zum Beispiel erkennen, ob der Benutzer des Telefons sich gerade in einem Auto befindet. Mit dem CO_4-Gehalt kann ermittelt werden, ob in der Nähe Alkohol oder eine ähnliche Substanz in der Luft ist. So kann möglicherweise ermittelt werden, ob der Benutzer sich gerade beim Feierabendbier in einem Restaurant befindet. Als fünfter und letzter Sensor ist das Board noch mit einem Temperatursensor[5] bestückt.



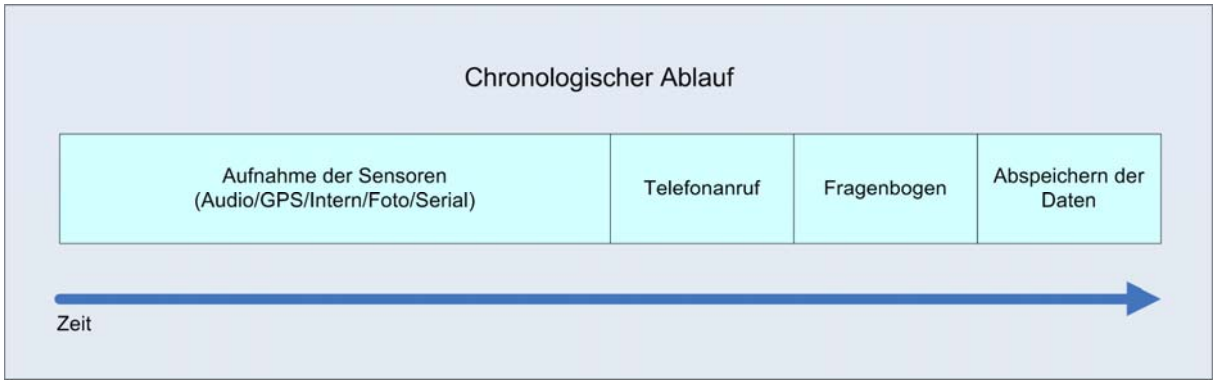
(Abb. 5: Sensorboard)

Um die externen Sensoren nochmals zusammenzufassen werden sie in der folgenden Aufzählung alle nochmals aufgelistet:

- GPS
- Sensorboard
  - Accelerometer [1]
  - Gyrosensoren [2]
  - Magnetfeldsensoren [3]
  - Gassensoren [4]
  - Temperatursensor [5]

3 Applikationsaufbau

Um den Versuch durchführen zu können und die Sensordaten zu dem jeweiligen Telefonanruf speichern zu können, muss eine Applikation erstellt werden, welche den folgenden Ablauf gewährleistet. Das Programm muss die ganze Zeit die Sensordaten aufnehmen und Sensordaten von etwa 10 Minuten in einem Buffer halten, so dass in dem Buffer die Sensordaten von 10 Minuten vor dem Anruf bis zu dem Anruf enthalten sind. Sobald ein Telefonanruf eingeht, muss dieser abgefangen werden um die Sensoraufnahme beenden zu können. Ebenfalls muss das Beenden des Anrufes abgefangen werden, damit anschliessend ein Menu zur Befragung des Benutzers gestartet werden kann. Sobald dieser Fragebogen ausgefüllt ist, muss sichergestellt werden, dass alle Sensordaten auf die SD-Karte gespeichert werden.
Die folgende Graphik illustriert den zeitlichen Ablauf des Programms:



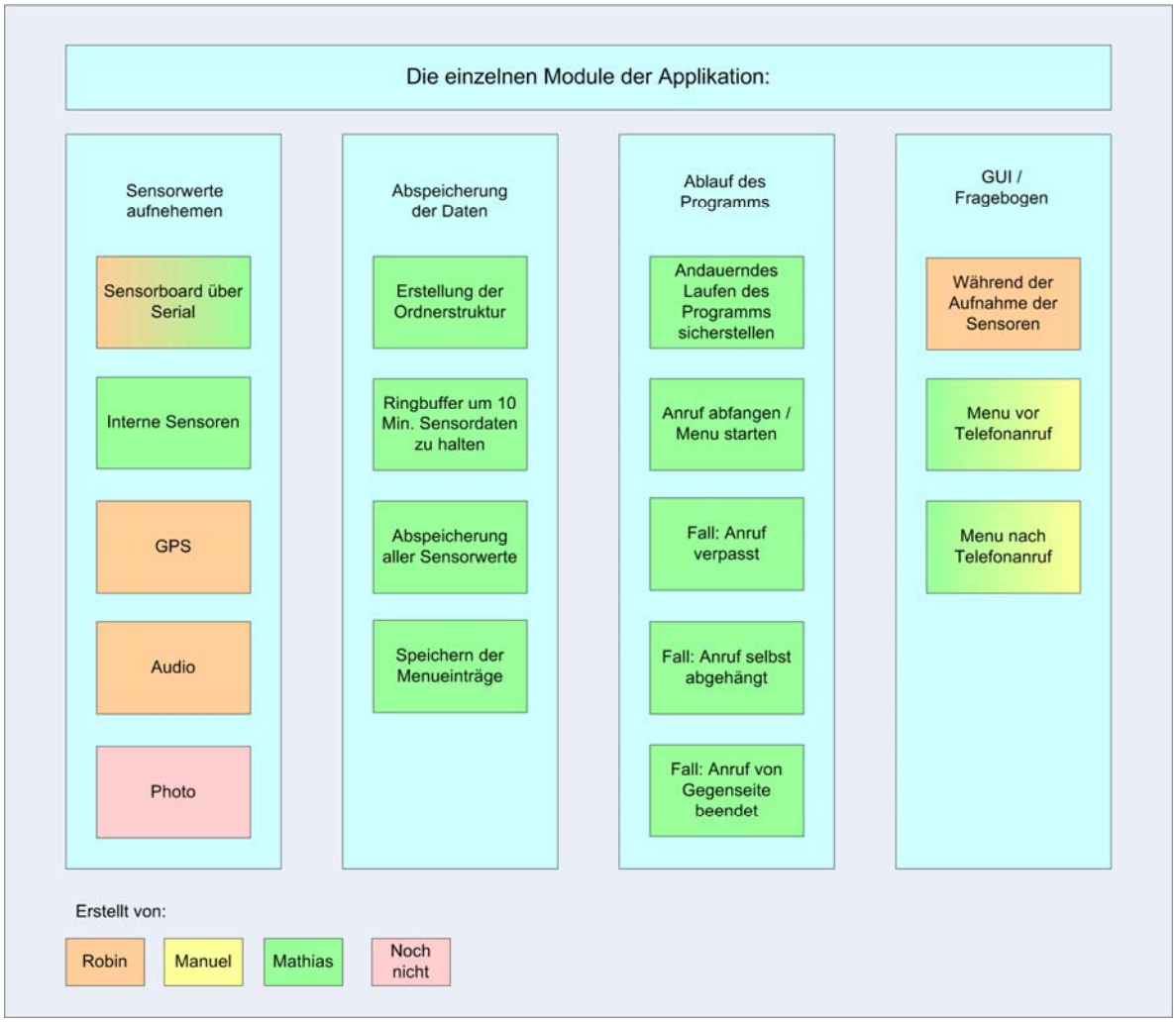
(Abb.6: Chronologischer Ablauf)

3.1 Module

Die Anforderungen lassen sich gliedern und in vier Grundmodule einteilen:

- Sensorwerte aufnehmen
- Abspeicherung der Daten
- Ablauf des Programms
- GUI /Fragebogen

Von meinen beiden Vorgängern, Robin Bucciarelli und Manuel Donner, wurden bereits einige Teile dieser Grundmodule bearbeitet. Mein Beitrag in dieser Diplomarbeit war es, die bereits bestehenden Teile zusammenzufügen, zu ergänzen und die fehlenden Module zu programmieren. Ich möchte kurz die Vorarbeit meiner Kommilitonen aufzeigen und auf die einzelnen Module eingehen. Im Implementierungs-Teil meiner Arbeit werde ich die Module noch genauer aufspalten sowie deren Umsetzungen, die aufgetretenen Probleme und deren Lösungen erläutern.



(Abb. 7: Module)

3.1.1 Sensorwerte aufnehmen

Die Integration aller Sensorstreams in eine einzige Applikation war das Ziel der Diplomarbeit von Robin Bucciarelli. Das Resultat dieser Diplomarbeit war ein Programm, welches gleichzeitig Audio, GPS und Daten des Sensorboardes über die serielle Schnittstelle aufnehmen kann. Es war Ihm nicht möglich, Photos zu machen und diese abzuspeichern und die aufgenommenen Daten über die serielle Schnittstelle sind so fehlerhaft, dass sie nicht verwendbar sind.[Bucciarelli, 2006]

Zu den bereits bestehenden Sensorstreams habe ich das Programm noch um interne Variabeln wie Signalstärke und Batteriestatus ergänzt.

3.1.2 GUI/Fragebogen

Manuel Donner hat sich in seiner Diplomarbeit unter anderem intensiv mit dem Design des Userinterfaces für den Fragebogen auseinander gesetzt. Das Benutzerinterface wurde so aufgebaut, dass es folgenden Kriterien genügt: Das Menu muss auf dem kleinen Bildschirm des Treo's deutlich sein und übersichtlich wirken, zusätzlich muss es schnell und intuitiv bedienbar sein. Das Menu ist sehr gut gelungen, was auch ein SUS-Test belegte. Es waren nur wenige Verbesserungen und Ergänzungen nötig. Hingegen fehlte dem Menu jegliche Funktionalität um die Werte des Fragebogens abzuspeichern.[Donner, 2006]

Die Menuführung wurde von mir optisch etwas verändert und um zwei Screens ergänzt. Auf die Veränderungen werde ich später noch genauer eingehen. Zudem habe ich das Menu in zwei eigenständige Menus aufgespaltet, damit eine Frage vor, und die restlichen nach dem Telefongespräch beantwortet werden müssen.

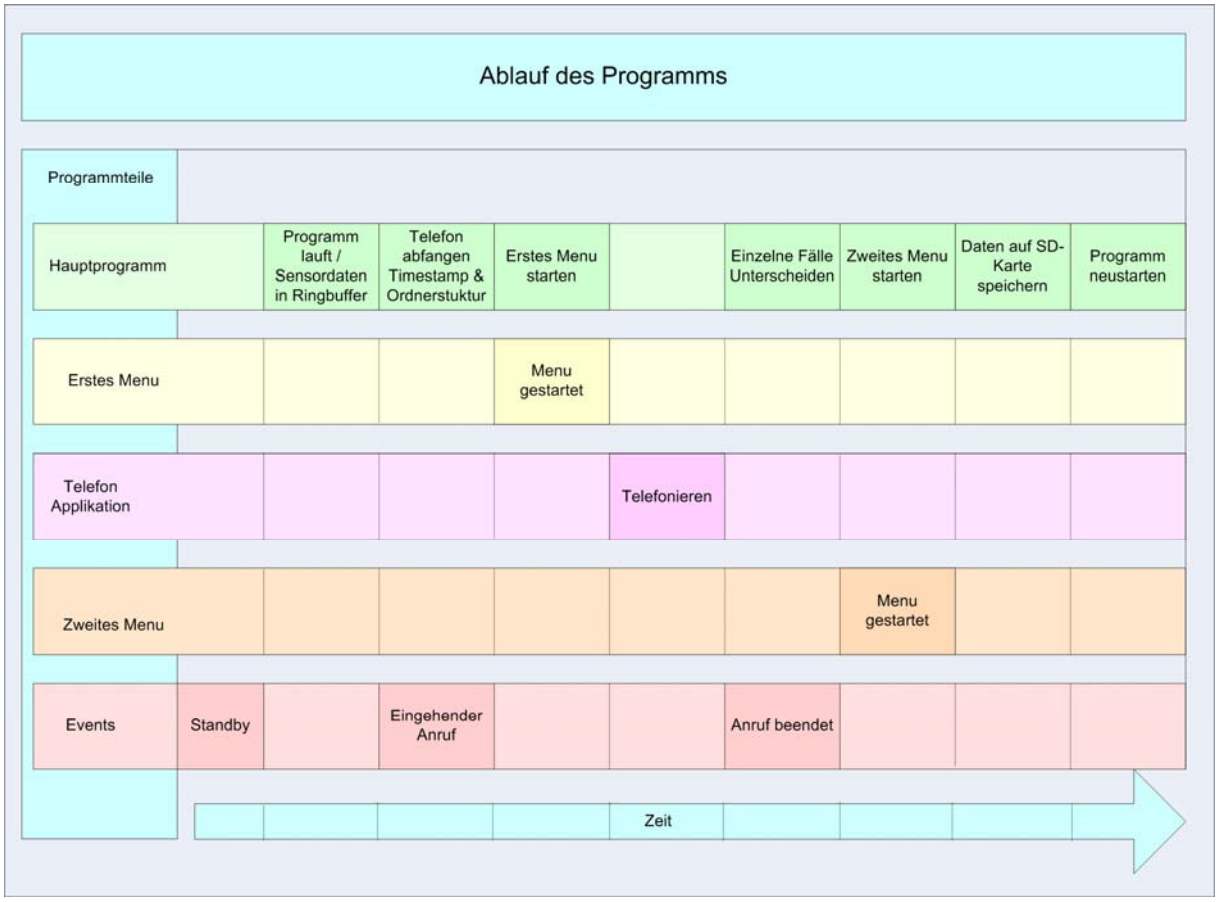
3.1.3 Abspeicherung der Daten

Um all die anfallenden Daten während des Experimentes abspeichern zu können, muss fortlaufend eine Ordnerstruktur erstellt werden um alle zu einem Telefonevent gehörenden Daten in einem separaten Ordner speichern zu können. Sobald ein Telefonanruf eingeht, müssen die vorhergehenden 10 Minuten abgespeichert werden können. Um dies zu ermöglichen, braucht es einen Ringbuffer, welcher die Sensordaten der 10 letzten Minuten halten kann.
Nachdem das Telefongespräch beendet und der Fragebogen ausgefüllt ist, müssen alle Sensordaten und die Werte des Fragebogens in das richtige Verzeichnis auf die SD-Karte abgespeichert werden.

3.1.4 Ablauf des Programms

Da das PalmOS 5.2 Garnet, welches als Betriebsystem auf dem Treo 650 läuft, nicht Multitasking fähig ist [PalmSource, 2006], ist dieses Modul wohl das komplexeste.
Das Ziel ist es, dass das Programm immer im Hintergrund die Sensordaten aufnimmt und das Telefon trotzdem noch vollumfänglich benutzbar ist. Dies ist so ohne Multitasking nicht möglich.
Da das Betriebsystem immer, wenn ein Event eintrifft, eine „Notification“ losschickt und da Multitasking, sofern nicht zwei oder mehre Prozessoren vorhanden sind, auch nur eine gesteuerte serielle Abarbeitung der Befehle ist [Stallings, 2003], lässt sich ein serieller, durch „Notificationen“ gesteuerter, Programmablauf konstruieren.

Sehr vereinfacht sieht der Programm Ablauf nun folgender massen aus:



(Abb. 8: Schematischer zeitlicher Ablauf des Programms)

Sobald das Telefon neu gestartet wird, eine Synchronisation abgeschlossen ist oder das Telefon versucht in den Standby-Modus zu gehen, wird dieser Event mittels „Notifacation“ abgefangen, das Programm zur Aufnahme der Sensordaten wird gestartet und die Sensordaten werden in einem Ringbuffer zwischengespeichert. Wenn der Benutzter nun eine andere Funktion des Telefons verwenden möchte, kann er dies problemlos tun. Sobald

er ein anderes Programm startet, wird das Aufnahmeprogramm beendet. Sobald das Telefon wieder in den „Standby-Modus“ wechseln möchte, wird wieder das Aufnahmeprogramm gestartet. Somit läuft das Programm immer, ausser es wird explizit vom Benutzer gestört.
Wenn ein Telefonanruf eingeht, wird von dem Programm die dementsprechende „Notification“ abgefangen, ein Ordner mit dem aktuellen Timestamp erstellt und der erste Teil des Fragebogenmenus gestartet. Sobald dieser beantwortet ist, kann telefoniert werden.
Ist das Gespräch beendet, muss diese Situation auch wieder abgefangen werden. Diese Situation kann aus drei verschieden Events bestehen: Entweder wurde der Anruf vom Benutzer (1) oder vom Anrufer (2) beendet oder der Anruf wurde verpasst (3). Das Abfangen dieser Situation erwies sich als ziemlich schwierig, da nicht für alle drei Fälle eine „Notification“ gesendet wird. Die API des Treo’s ist nicht dafür ausgelegt, in einem Programm Anrufe entgegenzunehmen, sondern eher dafür, aus einem Programm heraus eine Daten-Verbindung aufzubauen und so „Client mässig“ zu agieren.
Sobald der Anruf beendet ist, wird nun der zweite Teil des Fragebogenmenus gestartet. Sobald dieser ausgefüllt ist, werden die Werte des Fragebogens und alle Sensordaten auf die SD-Karte in das zu dem Telefonevent zugehörige Verzeichnis abgespeichert. Ist dies erledigt, wird das Programm von neuem gestartet, die Sensordaten werden wieder aufgenommen und der Ablauf beginnt wieder von vorne.

3.2 Einführung PalmOS

In diesem Kapitel möchte ich einige allgemeine Grundlagen zum Aufbau und zur Programmierung von PalmOS aufzeigen. Dabei werde ich speziell auf diejenigen Themen eingehen, welche von Bedeutung für das Erstellen der Applikation waren.

Beim Erstellen eines Programms für ein mobiles Kommunikationsgerät müssen viele Aspekte berücksichtigt werden.
Der beschränkte Befehlssatz der API von PalmSource und die dazu gehörigen Ergänzungen von PalmOne definiert exakt, was gemacht werden kann und was nicht. Was in einem normalen C-Progamm auf einem Computer gemacht werden kann, muss nicht unbedingt auf dem Telefon realisierbar sein. Zusätzlich können gewisse Systemfunktionen nur dann abgefragt oder verwendet werden, wenn dies explizit in der API ausprogrammiert worden ist.

Zudem ist es ein grosses Hindernis, dass nur ein kleiner Bildschirm zur Verfügung steht. Dies bringt eine weitere Designschwierigkeit mit sich. Man muss versuchen alle Funktionen in dem Menu unterzubringen, ohne den Bildschirm zu überladen oder unübersichtlich zu werden. [Shneiderman, 1997]

Man muss unbedingt berücksichtigen, dass man in einer sehr beschränkten Umgebung programmiert. Dies ist hauptsächlich spürbar im beschränkten Speicherplatz und Arbeitsspeicher. So muss sehr auf die verfügbaren Ressourcen geachtet werden. Dies ist wahrscheinlich auch ein Grund weswegen PalmOS sich entschieden hat ihr Betriebssystem auf C aufzubauen. Obwohl diese Programmiersprache schon etwas in die Jahre gekommen und durch das Pointerhandling etwas umständlich ist, ist sie sehr effektiv und Ressourcen sparend. Nicht ohne Grund dient C als Grundlage für moderne Programmiersprachen, wie z.B. Java.
Zu diesen Ressourcenknappheiten kommen aber auch noch die sehr eingeschränkten und erschwerten Debuggingmöglichkeiten.

3.2.1 Debugging

Das erstellte Programm kann zwar auf dem Computer mittels eines Emulators getestet werden, aber wenn es so funktioniert, heisst dies noch lange nicht, dass das Programm auch auf dem Telefon selbst läuft. Im Vergleich zu einem Debugger auf einem Computer, welcher bei einem Programmabsturz nachvollziehbar uns mitteilt, in welcher Datei, bei welcher Codezeile das Programm abgestürzt ist, startet das Telefon lediglich neu und man hat kaum einen Anhaltspunkt wo und warum sich der Absturz ereignet hat. Man kann zwar Fehler abfangen und die dazugehörigen Fehlermeldungen in einem Formular auf dem Bildschirm darstellen lassen, doch gibt es etliche Situationen, wie das Beenden des Programms oder während des Telefonierens, in welchen man nichts auf dem Display darstellen kann. Für diese Situationen haben sich „Systembeeps“ als sehr hilfreiches Mittel

erwiesen, da man so hört, ob ein Befehl ausgeführt worden ist oder ob das Telefon vorher neu startete. Ein „Systembeeb“ lässt sich mit folgendem Befehl ausführen:

```
SndPlaySystemSound(sndWarning);
```

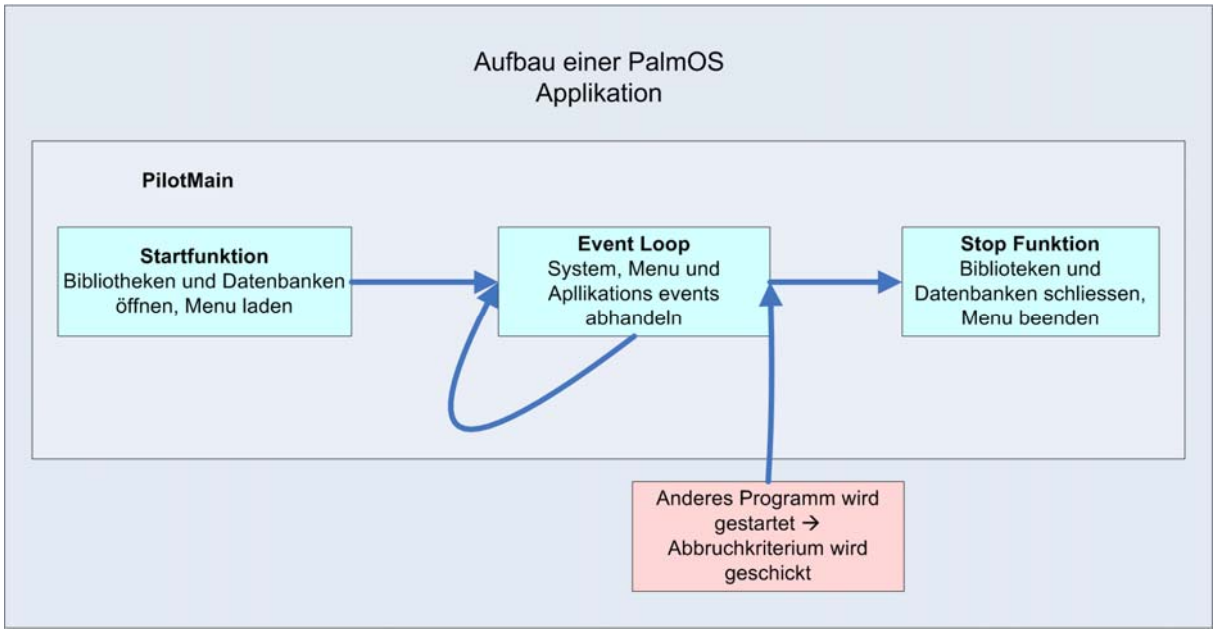
Im PalmOS ist ein eigener Debugger integriert. Es lässt sich einstellen, dass das Programm bei einem kritischen Fehler, anstatt abzustürzen in den „Debug-Modus“ wechseln soll. Das Problem ist nur, dass der Debugger auf Assembler-Ebene ist und die Meldungen des Debuggers unter Umständen sehr kompliziert zu interpretieren sind.
Zusätzlich speichert der Treo ein Systemfehlerprotokoll auf, kurz bevor das Telefon sich selbst neu startet. Dieses Systemfehlerprotokoll kann ausgelesen werden, indem in der Anrufansicht der Telefonapplikation die Kombination „#*377“ eingegeben und anschliessend auf die Anruftaste gedrückt wird.[Palm, 2006]

(Abb. 9: Systemfehlerprotokoll)

Dieses Systemfehlerprotokoll zeigt aber lediglich auf, welche Applikation zu welcher Zeit und welchem Datum abgestürzt ist und um was für eine Art von Fehler es sich handelte. Über die genauere Absturzursache, z.B. bei welcher Programmzeile oder bei welchem API-Aufruf sich der Fehler ereignet hat wird nicht protokolliert.

3.2.2 Aufbau einer Applikation

Der Einstiegspunkt des Compilers ist die Funktion mit dem Namen `PilotMain`. Mit dieser Funktion beginnt das Programm seriell die Funktionen abzuarbeiten. Üblicherweise enthält die Funktion `PilotMain`, in welcher wiederum drei weitere Funktionen enthalten sind.
Der standardmässige Aufbau wird in diesem Diagramm aufgezeigt:

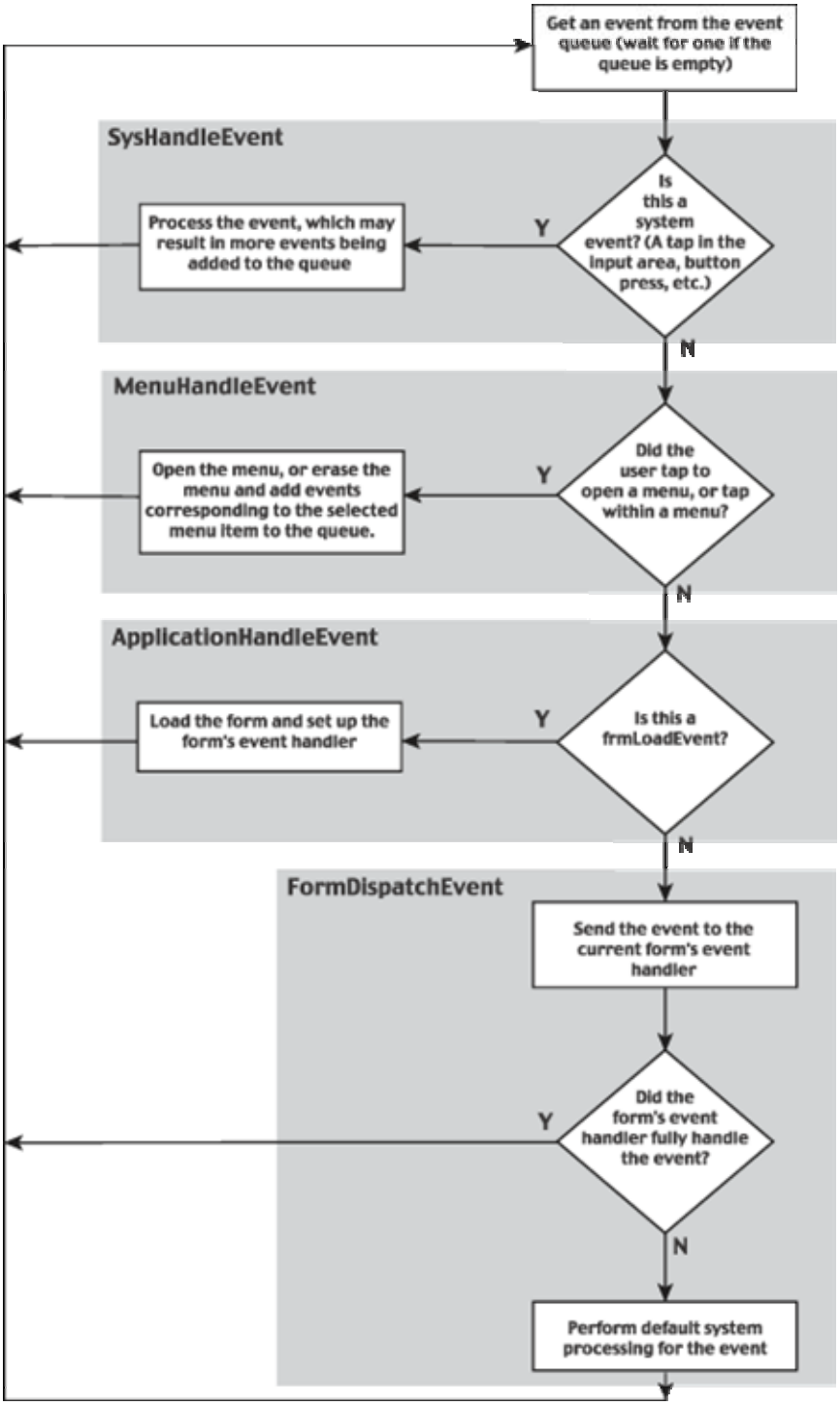


(Abb. 10: Aufbau einer PalmOS Applikation)

In der ersten, hier „Startfunktion“ genannt, werden alle Aufgaben abgearbeitet, welche zur Initialisierung des Programms notwendig sind, wie zum Beispiel das Öffnen von Systembibliotheken oder von Datenbanken. Am Ende dieser Funktion wird das Menu (GUI) der Applikation gestartet. Die Funktion „Event Loop“ ist das Herzstück der Applikation. Sie baut auf dem „Event Manager“ auf, welcher im nächsten Abschnitt noch genauer beschrieben wird. In dieser Funktion wird die gesamte Funktionalität des Programms zur Verfügung gestellt und es werden alle Events, wie zum Beispiel ein Tastendruck des Benützers, abgehandelt. Da PalmOS keine Möglichkeit vorsieht ein Programm zu beenden, ist dieser „EventLoop“ innerhalb einer Endlosschleife aufgebaut. Sobald eine andere Applikation gestartet werden soll, wird dem aktuell laufenden Programm ein Abbruchbefehl gesendet. Erst so wird die Endlosschleife beendet. Ist die Schleife abgebrochen, wird dem Programm noch mit der Funktion „Stop Funktion“ die Möglichkeit gelassen wichtige Sachen abzuschliessen, bevor die andere Applikation gestartet wird. In dieser Funktion werden normalerweise die verwendeten Systembibliotheken und Datenbanken wieder geschlossen, damit sie auch von anderen Programmen verwendet werden können. Hier wird auch das Menu wieder geschlossen. Erst wenn dies alles abgeschlossen ist, wird die neue Applikation gestartet. [PalmSource, 2006]

3.2.3 Eventmanager

Wie schon im letzten Abschnitt erwähnt, bildet der „Event Manger“ das Kernstück jeder PalmOS Applikation. Wie dieser Manager in der Endlosschleife die Events abhandelt und wie die Prioritätenverteilung erfolgt, lässt sich im folgenden Schema erkennen:



(Abb. 11: Event Manager, [PalmSource, 2006])

Dieser Event Manager wartet immer auf ein Ereignis, welches er abarbeiten kann. Solange kein Ereignis eintrifft, macht er rein gar nichts. Um aber die Funktionalität der Applikation

gewährleisten zu können, müssen fortlaufend (und nicht erst wenn ein Event eintrifft) Befehle zur Aufnahme von Sensoren ausgeführt werden können. Um so etwas zu realisieren, gibt es die Möglichkeit, dass der Event Manager in einem Zeitintervall (Systemtick oder ein Vielfaches jenes) immer wieder überprüft, ob ein Event ansteht. Ist kein Event abzuarbeiten, lässt sich ein `nilEvent` senden, welcher dazu verwendet werden kann Befehle auszuführen sobald der Event Manager nichts anders mehr zu tun hat.[PalmSource, 2006]

3.2.4 Applikationen starten

Um aus einem Programm ein anderes Programm zu starten bietet PalmOS zwei Möglichkeiten. Man kann mit dem API Aufruf `SysAppLaunch` ein Programm starten. Diese Methode startet das neue Programm Sobald dieses abgearbeitet ist kehrt es wieder zum ursprünglichen Programm zurück. Möchte man das aktuelle Programm ganz beenden und nur noch das neue Programm starten, kann man dies mit dem Befehl `SysUIAppSwitch` erreichen. Beiden Befehle können Parameter wie „Launchpriorität“ oder „Launchcodes“ mitgeliefert werden. So kann das neue Programm in einem bestimmten Modus, abhängig vom mitgeschickten „Launchcode“, gestartet werden.

3.2.4.1 Launchcodes

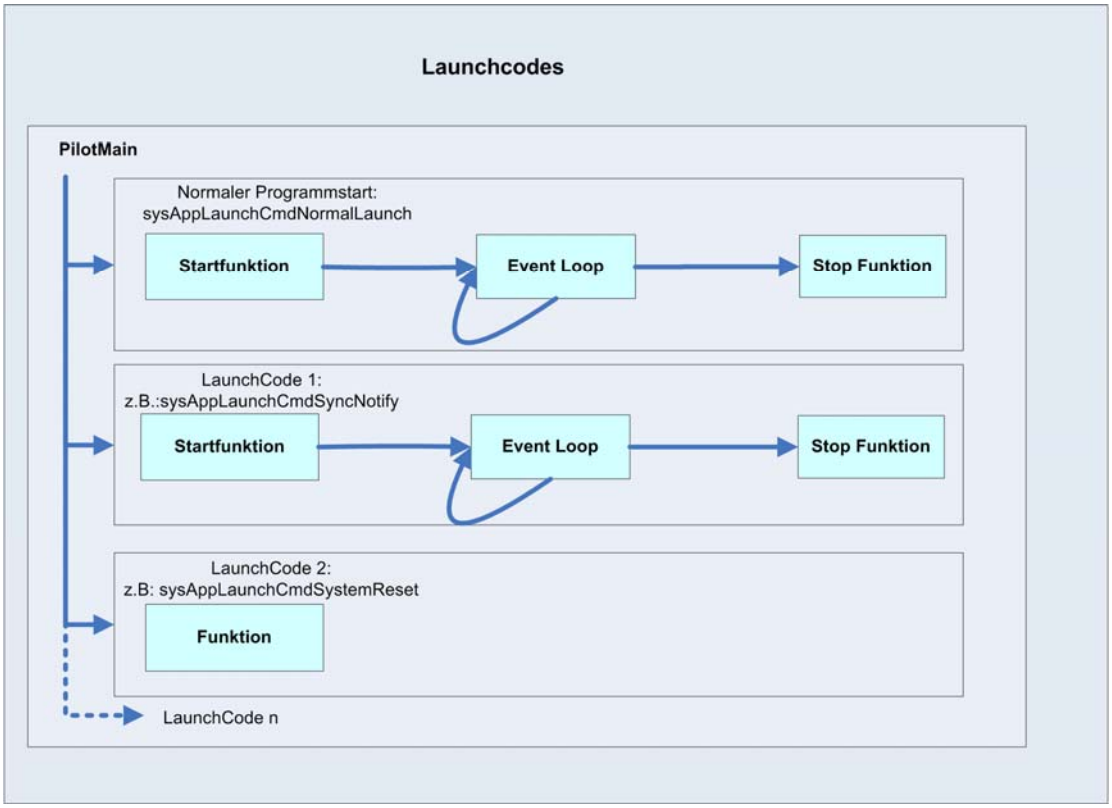
PalmOS bietet die Möglichkeit einer Applikation, welche gestartet wird, einen „Launchcode“ mitzugeben. Diese „Launchcodes“ ermöglichen, dass je nach Bedarf ein anderer Teil des Programms ausgeführt werden kann. Wenn ein Programm ohne „Launchcode“ gestartet wird, wird automatisch der Programmteil ausgeführt, welcher unter dem Standart „Launchcode“ `sysAppLaunchCmdNormalLaunch` steht.

Ich möchte dies kurz am Beispiel der Integrierten Telefonapplikation des Treo 650 erläutern.
Startet man die Telefonapplikation ohne „Launchcode“ so gelangt man in die normale „Home-Ansicht“ der Telefonapplikation. Möchte man nun aber die Telefonapplikation starten und gleich alle verpassten Anrufe anzeigen lassen, muss man den „Launchcode“ `phoneAppLaunchCmdViewMissedCalls` mitschicken. Oder man stellt sich vor, dass man aus einer eigenen Applikation eine vorgegebene Nummer anrufen möchte, so kann man beim Start der Telefonapplikation den „Launchcode“ `phoneAppLaunchCmdDial` und in einer Variabel die Telefonnummer übergeben. So startet die Telefonapplikation und wählt gleich die übergebene Nummer, ohne dass sonst noch etwas gemacht werden muss.

(Abb. 12: Home-Ansicht des Telefons)

(Abb. 13: Verpasste Anrufe Ansicht)

So wird je nach Launchcode nur kurz eine Funktion ausgeführt oder gerade in einen bestimmten Teil des Programms gesprungen.



(Abb. 14:Launchcodes)

Zusätzlich werden bei bestimmten Ereignissen, wie zum Beispiel das Beenden einer Synchronisation, in allen Programmen, welche eine Aktion für den entsprechenden Launchcode vorgesehen haben, die entsprechenden Programmteile ausgeführt. So kann auf gewisse Ereignisse entsprechend reagiert werden. Ein Beispiel hierfür ist, wenn eine Applikation die serielle Schnittstelle verwendet, ist diese besetzt und eine Synchronisation über diese Schnittstelle ist nicht möglich. Fängt man nun aber, mit einem entsprechenden

Launchcode (`sysAppLaunchCmdSyncNotify`), den Versuch eine Synchronisation zu starten, ab und schliesst die serielle Schnittstelle, ist diese nun für die Synchronisation verfügbar und kann ausgeführt werden.
Es gibt auch die Möglichkeit, selbst „Launchcodes“ zu definieren, auf welche man dann beim Aufrufen dieser Applikation zugreifen kann. Die im letzten Abschnitt beschriebene Funktion `SysAppLaunch` kann auch dazu verwendet werden um in dem momentan laufenden Programm in einen anderen Launchcode zu springen, indem man das Programm nochmals neu startet und den entsprechenden Launchcode mitliefert.

Es ist unbedingt zu beachten, dass zwischen den einzelnen Teilen der verschieden „Launchcodes“ kein Zugriff auf globale Variabeln anderer Teile möglich ist, da diese ganz getrennt ausgeführt werden.[PalmSource, 2006]

3.2.4.1 Notifications

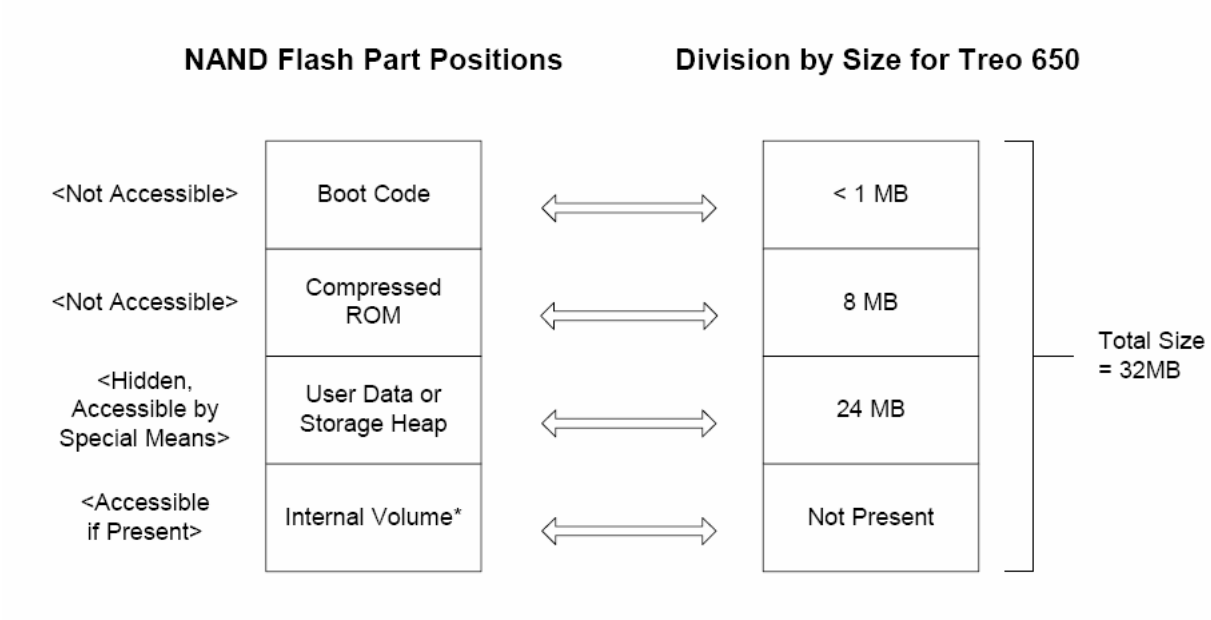
„Launchcodes“, welche auf bestimmte Systemereignisse gesendet werden, gibt es nur sehr wenige. Dies ist auch verständlich, denn wenn bei jedem x-beliebigen Event jedes Programm nach dem zugehörigen Codeteile abgesucht werden müsste, würde ein immenser „Overhead“ generiert. Um auf verschiedene Systemevents reagieren zu können, für welche kein eigener Launchcode existiert, gib es „Notifications“, zwar bei weitem nicht für alle doch für die wichtigsten Ereignisse sind „Notifications“ implementiert. „Notifications“ werden im Unterschied zu den „Systemlaunchcodes“ nur an die Programme geschickt, welche sich zuvor für diese bestimmte “Notification“ registriert haben.
Das Registrieren für eine „Notification“ erfolgt durch folgenden Aufruf, wobei für *`Notificationsname`* der entsprechende Name der gewünschten „Notification“ einzusetzen ist. Eine Liste aller verfügbaren „Notificationen“ findet sich im Anhang.

```
SysNotifyRegister(cardNo, dbID, Notificationsname, NULL,
sysNotifyNormalPriority, 0);
```

Das Abarbeiten einer „Notification“ läuft nun wieder identisch zu den „Launchcodes“. Ist ein Programm für eine oder mehre „Notifications“ registriert, wird das betreffende Programm nun gestartet. Falls es schon läuft, wird es neu gestartet mit dem „Launchcode“ `sysAppLaunchCmdNotify`. Zusätzlich wird ein Parameter übergeben, aus welchem sich die Art der „Notification“ auslesen lässt. So ist es möglich auf verschiedene „Notificationen“ spezifisch zu reagieren.
Nach einem „Systemreset“, egal ob Hard- oder Soft- Reset, ist das Programm nicht mehr für die „Notification“ registriert. Ebenso, wenn das Programm gelöscht wird, wird automatisch die Registrierung für die „Notification“ wieder aufgehoben. Sollte es jedoch nötig sein, die Registrierung einer „Notification“ wieder aufzuheben, ist dies mit dem folgenden Befehl möglich: `SysNotifyUnregister(cardNo, dbID, Notificationname, 0);` [PalmSource, 2006]

3.2.5 Memory

Wie schon in der Einleitung dieses Kapitels erwähnt wurde, müssen die beschränkten Ressourcen während des Programmierens beachtet werden. Da in den 10 Minuten Aufnahmezeit einiges an Sensordaten zusammenkommt, welche im Speicher gehalten werden müssen, bis sie bei einem Telefonevent auf die SD-Karte abgespeichert werden können, möchte ich auf den verfügbaren Speicher des Treo's eingehen.
Der Treo verfügt über einen 32MB grossen NAND Flash[3] Speicher und über einen 32MB grossen Arbeitsspeicher (RAM)
Der NAND Speicher ist folgendermassen aufgeteilt:



(Abb. 15: NAND Speicher,[Palm, 2006])

Etwa ein MB wird von dem Boot Code belegt, in welchem die Informationen zum Aufstarten des Betriebsystems enthalten sind. 8 MB werden von dem komprimierten ROM belegt, welches alle Programme bei Auslieferung des Treo's enthält und welche auch nach einem „Hardreset" wieder hergestellt werden. Diese beiden Teile sind nicht zugänglich. Die restlichen 24 MB sind für neu installierte Programme und Benutzerdaten, wie Terminkalender etc. reserviert.

Die 32 MB RAM stehen dem Programmierer aber nicht direkt zur Verfügung. Sie werden folgendermassen bereits verwendet:

[3] Die **NAND**-Verknüpfung (engl. **n**ot **and** = de. nicht und;) ist in der Informatik und der Aussagenlogik ein boolescher Operator bzw. Junktor, der die Negation des Verknüpfens zweier boolescher Variablen durch die Konjunktion (AND-Verknüpfung) darstellt. Die Gesamtaussage zweier durch die NAND-Verknüpfung verknüpften Aussagen ist also wahr, wenn mindestens eine Aussage falsch ist, bzw. dann falsch, wenn beide wahr sind.[Wikipedia,2006]

(Abb. 16: RAM Partitionen, [PalmSource, 2006])

16 MB werden sofort nach dem Systemstart belegt indem das komprimierte ROM aus dem NAND-Speicher dekomprimiert wird und für den Gebrauch in den Arbeitsspeicher gelegt wird. Weiter 10 MB sind für den Gebrauch von Datenbanken wie Adressbücher oder Terminkalender reserviert. So bleiben für den Betrieb nur noch 6MB.
Die extremste Einschränkung kommt aber nicht durch die beschränkte Arbeitsspeichergrösse, sondern durch die Designentscheidung von PalmSource, dass alle Programme, welche erstellt werden abwärtskompatibel zu ältern Geräten sein sollten. Diese Abwärtskompatibilität ist vollkommen illusorisch, da spezielle Funktionen bei jedem Palm Model je nach Hersteller andere API Aufrufe verwenden und die Geräte ganz andere Systembibliotheken besitzen.
Dennoch ist im PalmOS 5.2 Garnet die maximale Grösse, wie viel Arbeitsspeicher eine Applikation verwenden darf, auf 68KB beschränkt.

3.3 Entwicklungsumgebung

Um für den Treo eine Applikation entwickeln zu können, braucht es einen SDK und eine Entwicklungsumgebung. Den SDK lässt sich auf der PalmSource Webpage unter der Rubrik „Developers“ herunterladen. Auf dieser Seite findet sich ebenfalls die API-Dokumentation, in welcher alle für den Palm verfügbaren Befehle aufgeführt sind, sowie einige „Developer’s Guides“ die für den Einstieg in die Palmprogrammierung sehr wertvoll sind. [PalmSource, 2006] Da der Treo aber nicht von PalmSource selbst entwickelt worden ist, sondern von PalmOne, funktionieren nicht alle Befehle, die in der API aufgeführt sind. Zusätzlich wurde von PalmOne einige Veränderungen vorgenommen und einiges spezifisch für den Treo ergänzt. Daher muss von der PalmOne Webpage noch ein Zusatz SDK herunter geladen und installiert werden. Auf dieser Seite findet sich ebenfalls eine API- Dokumentation,welche zwar sehr dürftig ausgefallen ist, und einige brauchbare Sampelcodes. So muss man die API von PalmSource als Grundlage verwenden, aber für speziellere oder für treospezifische Funktionen, vor allem in Zusammenhang mit den Telefonfunktionen, auf die API von PalmOne zurückgreifen.[Palm, 2006]

Als Entwicklungsumgebung fiel die Wahl auf den Codewarrior 9.3 für Palm. Die Entscheidung war bedingt, dadurch, dass alle meine Vorgänger schon mit dem Coderwarrior gearbeitet haben und mir ihr Wissen vermitteln konnten und da schon Projekte für den Codewarrior vorhanden waren. Negativ aufgefallen bei Codewarrior ist, dass der Linker nicht richtig funktionierte. Als Alternative wird auf der PalmSource Seite eine kostenlose Entwicklungsumgebung angeboten. Diese heisst Palm OS Developer Suite und baut auf Eclipse auf.

4 Implementierung

In diesem Teil meiner Diplomarbeit werde ich auf die konkrete Umsetzung der im Kapitel 3.1 erwähnten Module eingehen, die aufgetretenen Probleme und deren Lösungen aufzeigen. Hierzu möchte ich noch erwähnen, dass das Hauptziel in der Umsetzung des Programms die Gewährleistung der erwünschten Funktionalitäten war. Da dieses Programm nur zur Durchführung des Experimentes verwendet wird und somit sein Lebenszyklus auf die Dauer des Experimentes beschränkt ist, wurde die Priorität der Entwicklung auf die Realisierung der Funktionalitäten gelegt. Hinzu kommt auch noch der Aspekt, dass bald der Treo 700 erscheinen wird, welcher unter Windows Mobil laufen wird und so eine ganz andere Programmierumgebung (C#) bietet, welche Multitasking und Multithreading erlaubt.
Der Treo 650 hingegen bietet nur eine sehr beschränkte Multithreadfähigkeit. So ist es nur möglich, speziell dafür vorgesehen Funktionen im Hintergrund ablaufen zu lassen, wie z.B. das Abspielen eines Klingeltones.

Um alle Anforderungen an das Programm richtig umsetzen zu können, habe ich fortlaufend eine Liste geführt, in welcher ich alle Ansprüche an das Programm und die möglichen Situation aufgelistet, überprüft und abgearbeitet habe. Diese Liste der „Requirements" findet sich im Anhang.

4.1 GUI/Fragebogen

Wie schon erwähnt, konnte ein grosser Teil des „Graphical User Interfaces" (GUI) von meinen beiden Vorgängern Manuel Donner und Robin Bucciarelli übernommen werden.

4.1.1 GUI des Sensoraufnahmeprogramms

Während der Sensoraufnahme ist Robins GUI bis jetzt weiter verwendet worden. Dieses GUI ist funktional orientiert entwickelt worden. Diese zeigt sich dadurch, dass auf einen Blick ersichtlich ist, welche „Sensorstreams", aufgenommen werden und dadurch, dass das ganze Programm über diesen Screen gesteuert werden kann. Bevor das Experiment gestartet wird, müssen bei diesem GUI noch die Buttons entfernt werden, damit der Benutzer nicht die Aufnahmen unterbrechen kann.

(Abb. 17: GUI Sensoraufnahme)

4.1.2 GUI des ersten Menus

Das von Manuel entworfene GUI für die Befragung des Benutzers zu dem Anruf, ist eine rein graphische Gestaltung, d.h. es fehlt dem Menu an jeglicher Funktionalität, wie z.B. das Speichern der Antworten.
Dieses Menu wurde in zwei einzelne Menus geteilt. Das erste Menu erscheint, sobald ein eingehender Anruf erkannt wird. Erst wenn dieses Menu ausgefüllt ist, ist es dem Benutzer möglich das Gespräch anzunehmen.
Um die Benutzerfreundlichkeit nicht allzu stark einzuschränken, wurde dieses Menu so kurz wie möglich gehalten. Es besteht lediglich aus der Frage, wie störend der Anruf für den Benutzer selbst war. So sollte die Zeit gut reichen, um die Frage zu beantworten und noch rechtzeitig das Telefon abnehmen zu können. Es wäre auch möglich gewesen, diese Frage erst nach dem Gespräch zu stellen, doch wäre z.B. nach einem langen Gespräch die Antwort unter Umständen verfälscht, da die Störung nicht mehr so präsent wäre.
Dem GUI von Manuel wurde noch eine Hintergrundfarbe hinzugefügt, damit besser ersichtlich ist, wo sich die Buttons befinden und zusätzlich wurde die Auflösung verfeinert.

(Abb. 18: GUI erstes Menu von Manuel)

(Abb. 19: GUI erstes Menu neu)

4.1.3 GUI des zweiten Menus

Der zweite Teil des Fragebogens wird erst nach dem Anruf gestartet, damit der Benutzer genügend Zeit hat alle Fragen korrekt zu beantworten.
Für das zweite Menu wurde Manuels GUI nur geringfügig angepasst. Einerseits wurde die Symbolik vereinfacht und klarer dargestellt anderseits wurde die Position der Buttons so überarbeit, dass auf zwei aufeinander folgenden Screens die Buttons nie an derselben Stelle sind. So wird vermieden, dass der Benutzer aus Versehen zweimal drückt und so das nächste Menu unbewusst ausfüllt.

(Abb. 20: Symbolik Manuel)

(Abb. 21: Symbolik neu)

(Abb. 22: Anordnung Manuel)

(Abb.23: Anordnung neu)

Sonst wurden alle Screens übernommen. Der Inhalt und Sinn dieser Fragen ist in Manuel Donners Diplomarbeit genauer erklärt[Donner, 2006].

4.1.4 Kostenerfassungs-Screen

Die Screens von Manuel wurden noch um einen weiteren ergänzt. Mit diesem Screen wird der Benutzer gefragt, wie gravierend es wäre, wenn das Telefon anders gehandelt hätte, als es vom Benutzer erwünscht war.
So kann durch einen Vergleich ermittelt werden, dass z.B. ein wichtiger Anruf, welcher fälschlicherweise durch das Telefon abgewiesen wurde, viel gravierender ist, als wenn das Telefon einen belanglosen Anruf in einem nicht so günstigen Moment durchgestellt hat. Dies lässt zu, dass die Kosten eines Fehlers des Telefons bzw. einer Missklassifikation erfasst werden und ermöglicht so eine kostensensitive Voraussage. Dieser Sachverhalt lässt sich dann in einer ROC-Kurve darstellen [Frank, 2005].

(Abb. 24: Kostenerfassungs-Screen)

4.1.5 Funktionalität des Menus

Die angepassten Menus von Manuel Donner mussten noch in das Programm integriert werden und um die Funktion, die Resultate der Befragung auch abspeichern zu können, erweitert werden. Der genaue Speichervorgang wird im Kapitel 4.33 und 4.4.4 beschrieben.
Eine weitere Designfrage ist, unter welchen Umständen der Fragebogen ausgefüllt werden muss und unter welchen der Fragebogen abgebrochen werden kann. Geht während dem Ausfüllen des zweiten Menus nach einem Anruf ein weiterer Anruf ein, wird das Menu übersprungen und der Anruf kann entgegengenommen werden. So geht zwar ein Event ganz verloren, doch würde es die Benutzerakzeptanz zu stark einschränken, wenn zuerst der Fragebogen fertig ausgefüllt werden müsste, und man so Gefahr läuft einen wichtigen Anruf zu verpassen.
In jedem anderen Fall bleibt das zweite Menu solange geöffnet, bis es fertig ausgefüllt wurde oder die Batterie des Treo's leer ist. Damit das Menu nicht abgebrochen werden kann, werden während dem Menu alle Hardwaretasten des Treo's deaktiviert und sobald das Menu fertig ausgefüllt ist, wieder aktiviert.

```
//Hardwaretasten deaktivieren
//Namen der Tasten finden sich in der Headerdatei HsKeyCodes.h
         HsKeyEnableKey  (keyHard1, false);
         HsKeyEnableKey  (keyHard2, false);
         HsKeyEnableKey  (keyHard3, false);
         HsKeyEnableKey  (keyHard4, false);

         HsKeyEnableKey  (keyLaunch, false);
         HsKeyEnableKey  (keyMenu, false);
```

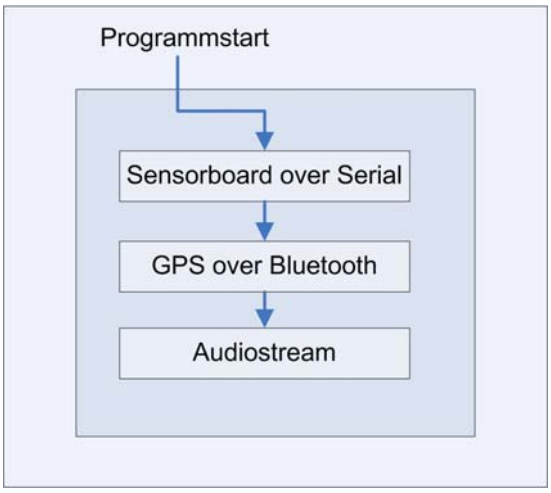
4.2 Sensorwerte aufnehmen

Der Sensorrecorder, welcher Robin Bucciarelli während seiner Diplomarbeit programmiert hat [Bucciarelli, 2006], musste in die Programmarchitektur des gesamten Programms integriert, angepasst und erweitert werden, damit das Programm den Anforderungen entspricht, welche die Voraussetzung für das geplante Experiment sind.
Die Hintergründe zu der seriellen Schnittstelle, Bluetooth und GPS wurden bereits in der Diplomarbeit von Robin Bucciarelli aufgezeigt [Bucciarelli, 2006] und werden hier nicht nochmals erläutert.

4.2.1 Sensoraufnahme starten

Sobald das Programm gestartet wird, muss die Aufnahme der Sensoren automatisch beginnen, damit der Benutzer des Telefons so wenig wie möglich selbst machen muss.

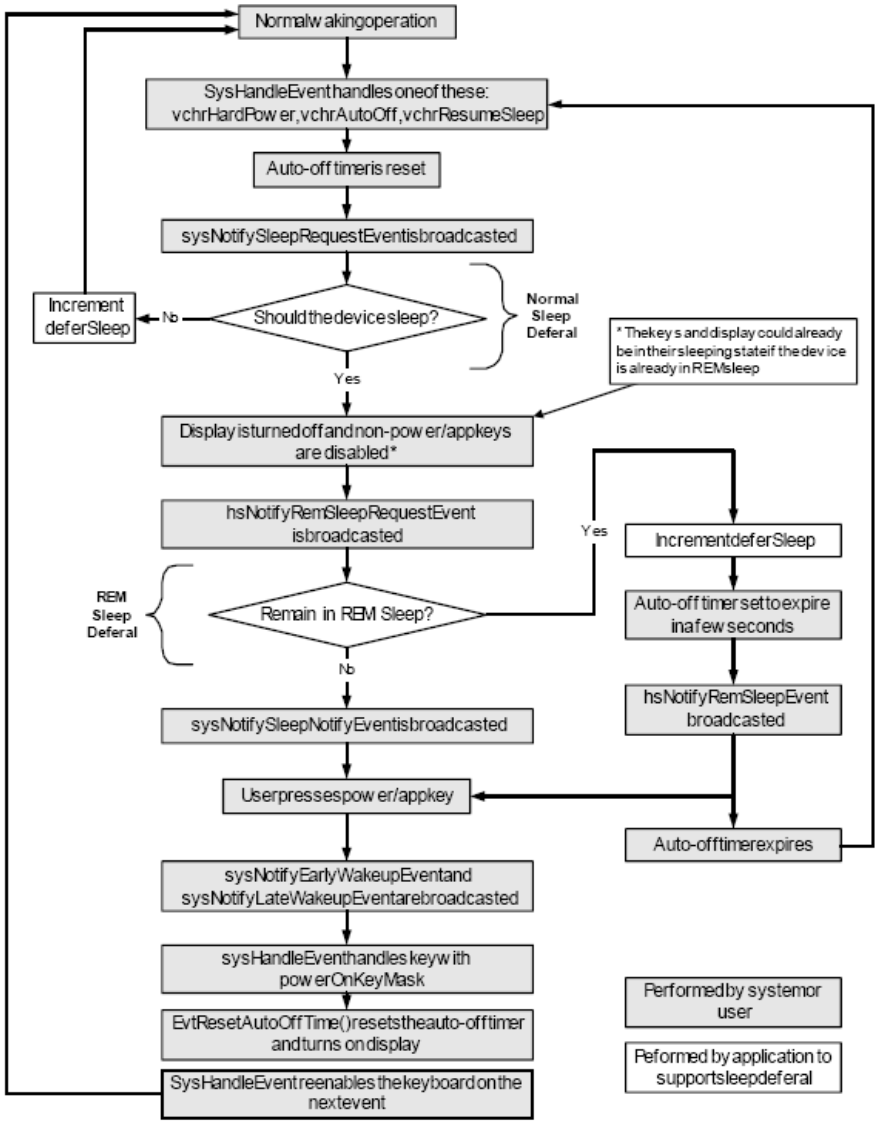
Damit die Sensoraufnahme reibungslos starten kann, ist es wichtig, dass folgende Reihenfolge exakt eingehalten wird.



(Abb. 25: Reihenfolge bei Programmstart)

Ist das Programm gestartet wird als erstes der serielle Port gesucht. Sobald dieser gefunden ist, wird er geöffnet und das Programm ist bereit über die serielle Schnittstelle Daten zu empfangen oder zu senden. Die Sensorboards von Peter Vorburger wurden so verändert, dass alle Sensoren, bis auf den Gassensor, standardmässig eingeschaltet sind. So muss dem Sensorboard der Befehl zu Aktivierung der Sensoren nicht mehr geschickt werden. Die Gassensoren werden wegen ihres hohen Energieverbrauches nur sporadisch eingeschaltet.
Als nächstes wird die Bluetooth Bibliothek geöffnet, damit das Programm für Rückrufe eines GPS-Gerätes registriert werden kann. Danach wird nach verfügbaren Bluetoothgeräten gesucht und sobald das GPS Modul gefunden ist, wird das GPS-Gerät über die Verbindung gekoppelt, so dass die gesendeten GPS-Daten von dem Programm empfangen werden können.
Als letztes wird die Audioaufnahme gestartet. Für diesen Zweck wird auf der SD-Karte eine Datei erstellt und geöffnet. In diese Datei wird ein „Waveheader“ geschrieben und ein Stream verbunden.
Sobald dies alles erledigt ist kann die Aufnahme beginnen. Der Stream zur Audioaufnahme wird gestartet, die GPS Daten werden über Bluetooth empfangen und über die serielle Schnittstelle werden die Daten des Sensorboardes empfangen.

Um sicherzustellen, dass das Programm zur Sensoraufnahme immer läuft, bzw. wieder neu gestartet wird, nachdem das Telefon für etwas anders benutzt worden ist, wird, sobald der Treo in den Standby gehen möchte, das Programm zur Sensoraufnahme gestartet. Wie dies funktioniert wird im Kapitel 4.4.1 noch genauer erklärt. An diesem Punkt ist es wichtig, dass der Versuch, in den „Standby-Modus“ zu gehen, hinausgeschoben werden muss, sobald das Programm gestartet wird. Wenn das Programm, sofort in den „Standby-Modus“ gehen würde, wäre zu wenig Zeit vorhanden, all die Schritte durchzuführen, welche nötig sind um die Sensoraufnahme zu starten.
Der Treo kennt drei Zustände: „Normaler Wachzustand“, „Versuch zu schlafen“ und eine „REM-Tiefschlaf-Phase“[4].
Es lässt sich mit dem Befehl `EvtResetAutoOffTimer()` verhindern, dass der Treo in den „Standby-Modus“ wechselt. Sobald dieser API-Aufruf getätigt wird, dauert es wieder 30 Sekunden, bis der Treo erneut versucht in den „Standby-Modus“ zu wechseln.
Das folgende Diagramm zeigt diese Zustände und die dazugehörigen Übergänge auf.



(Abb. 26: Sleep Diagramm)

[4] Wenn in dieser Arbeit von „Standby-Modus“ gesprochen wird ist immer „Versuch zu schlafen gemeint“.

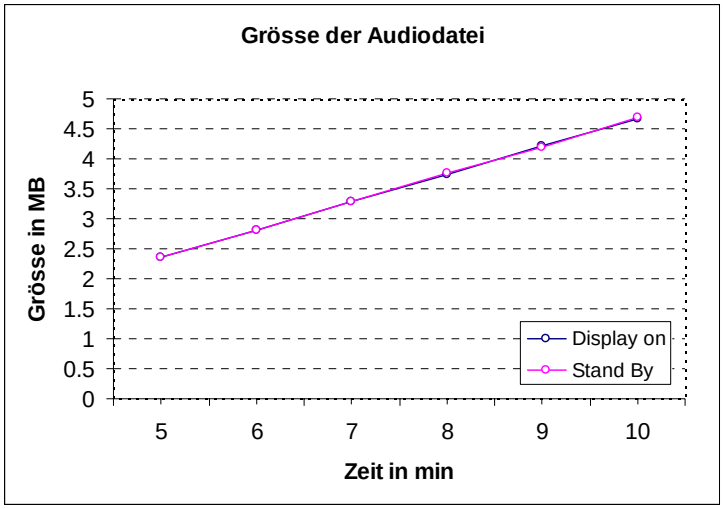
Während der Aufnahme kommt der Treo nie ganz in die „REM-Sleep“ Phase. Er schaltet zwar das Display aus und sperrt die Tasten, aber durch die Aufnahmen wird er immer wieder beschäftigt, und kann so nie ganz in den „REM-Sleep“ Modus gehen. Dies erklärt, nebst dem Stromverbrauch des Sensorboards, den in Robin Bucciarelli’s Diplomarbeit [Bucciarelli, 2006] gemessen, relativ hohen, Energieverbrauch während den Aufnahmen. So reicht der Akku des Treo’s für ca. 9h Aufnahmezeit.
Für das Beenden des Aufnahmeprogramms muss, wie beim Aufstarten, die gleiche Reihenfolge eingehalten werden. So muss zuerst der serielle Port, dann die Bluetooth Verbindung und zuletzt der „Audiostream“ geschlossen werden.
In den folgenden Kapiteln wird auf die einzelnen Sensorgruppen noch genauer eingegangen und aufgezeigt was beachtet, verändert und ergänzt werden musste.

Um sicher zu gehen, dass auch während das Programm im „Standby-Modus“ ist, genügend Daten aufgenommen wurden, ist für jede Sensorgruppe eine Vergleichsmessung[5] gemacht worden, in welcher im „Standby-Modus“ und bei eingeschaltetem Display aufgenommen und diese Aufnahmen miteinander verglichen wurden. Aus dieser Messung lässt sich auch erkennen, wie gross der anfallende Speicherbedarf für die Sensoraufnahmen ist.

4.2.2 Audiostream

Die Aufnahme des „Audiostreams“ konnte fast eins zu eins von Robins Programm übernommen werden. Es musste lediglich um eine Bufferfunktion ergänzt werden, welche im Kapitel 4.3.2.3 „ Ringbuffer für Audio“ noch genauer erläutert wird.
Die Audioaufnahme ist die einzige Sensordatenaufnahmefunktion, welche in den Hintergrund geschickt und parallel zu anderen Funktionen ausgeführt werden kann.

Für die Audiodatei wurde die Grösse gemessen und in folgendem Diagramm dargestellt:

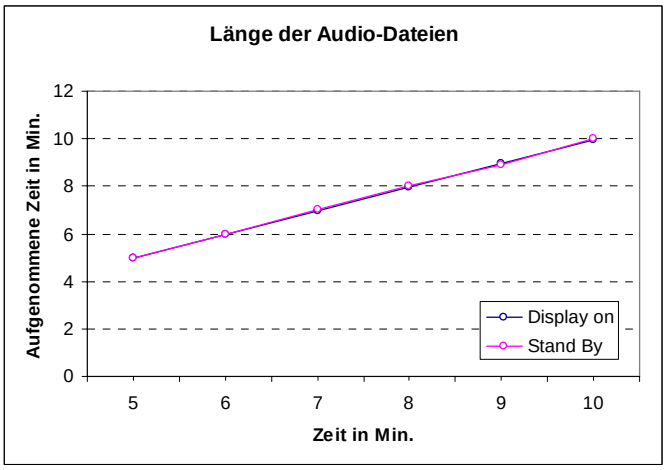


(Abb. 27: Grösse der Audiodatei)

Aus diesem Diagramm ist ersichtlich, dass die aufgenommene Datenmenge während dem „Standby-Modus“ und während dem das Display aktiviert war, in etwa identisch ist.

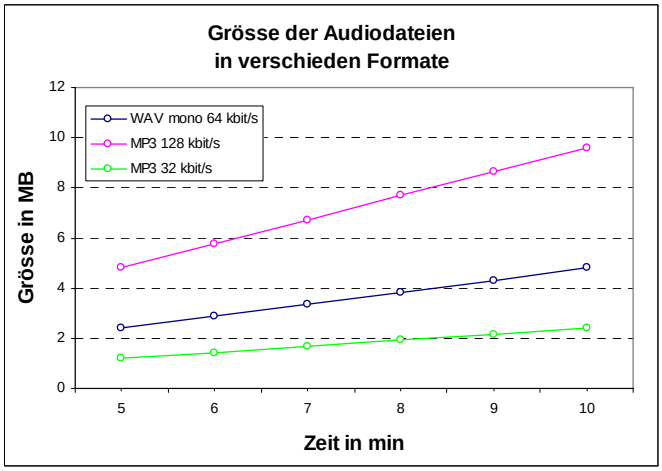
[5] Es wurde für jeden Zeitpunkt eine einzelne und unabhängige Messung gemacht.

Dies bestätigt sich zusätzlich durch die Messung der effektiven Zeitdauer[6] der Aufnahme, welche in folgendem Diagramm dargestellt ist:



(Abb. 28: Länge der Aufnahme)

Es gibt noch die Möglichkeit, diese Audiodatei, welche im WAV-Format gespeichert ist, mit dem Treo internen Codec in eine MP3-Datei zu konvertieren. Die dazu benötigten Funktionen finden sich in der Headerdatei „`palmOneCodecFormat.h`“, welche ein Teil der Erweiterungen von PalmOne ist. All die Formate welche von diesem Codecmanager hin und her konvertiert werden können finden sich in der Headerdatei „`palmOneCodecFormat.h`“. So werden nebst Audioformaten auch noch verschiedene Bild- und Videoformate unterstützt.
Würde man die Audiodateien im MP3Format mit 32 Kbits/s speichern, könnte der benötigte Speicherplatz halbiert werden. Der Platzbedarf für die verschieden Formate ist im folgenden Diagramm dargestellt.



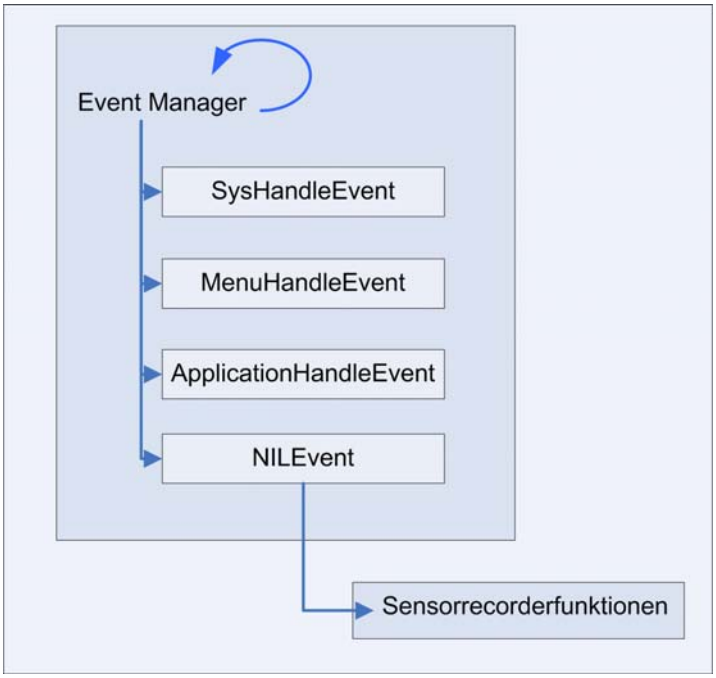
(Abb. 29: Grösse der Audiodatei in Verschieden Formaten)

Es ist aber noch abzuklären, ob es möglich ist, die Konvertierung sich in den „Audiostream“ integrieren lässt und im Hintergrund ablaufen kann. Falls das nicht realisierbar ist, könnte immer noch die Konvertierung nach einem Event durchgeführt werden, während dem die restlichen Sensordaten gespeichert werden.

[6] Mit dieser Messung wird überprüft, ob in z.B. 5 Minuten Aufnahmezeitwirklich eine 5 Minuten lange Wavedatei entsteht. Dies wird benötigt, um zu zeigen, dass auch während dem „Standby-Modus“ eine lückenlose Aufnahme statfindet.

4.2.3 GPS / Bluetooth

Das Empfangen der GPS-Daten über die Bluetooth Schnittstelle wurde ebenfalls aus Robins Programm übernommen. Bei der Integration dieses Programmteils, musste im Speziellen darauf geachtet werden, dass bei einem Beenden des Programms, egal ob durch den Benutzer oder durch einen Anruf, die Verbindung zu dem Bluetooth GPS-Empfänger sauber getrennt wird, da dies sonst zu schwer nachvollziehbaren Implikationen und Abstürzen führen kann.
Da diese Funktion nicht in den Hintergrund geschickt werden kann, muss sie nebst den anderen Sensoraufnahmefunktionen der Reihe nach, seriell abgehandelt werden. Die Umsetzung sieht folgendermassen aus: Sobald im Event Manager kein abzuhandelnder Event ansteht sendet dieser einen `NILEvent`, welcher abgefangen und ausgenützt werden kann, um die Funktionen zur Sensoraufnahme der Reihe nach durch zu gehen.



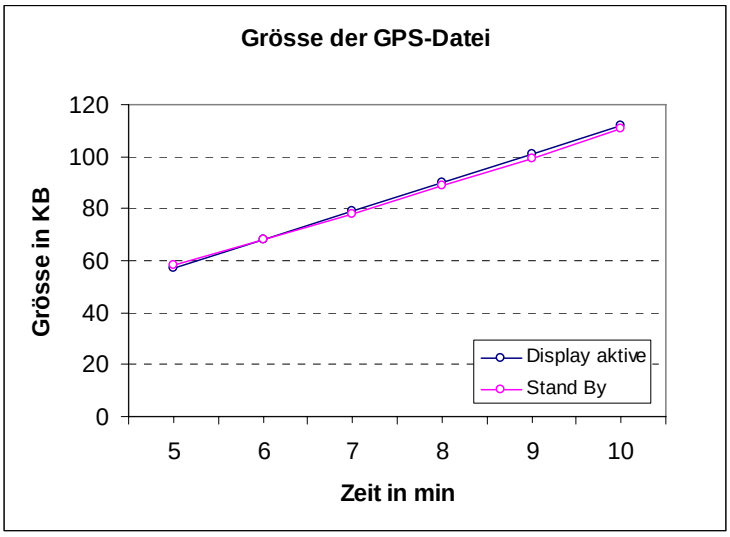
(Abb. 30: Ablauf Sensoraufnahme)

Die aufgenommen GPS-Daten sehen nun folgendermassen aus:

```
…
$GPGGA,235948.000,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*77
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05210
$GPRMC,235948.000,V,0000.0000,N,00000.0000,E,0.000000,,091102,,*00
$GPGGA,235949.000,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*76
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,235949.000,V,0000.0000,N,00000.0000,E,0.000000,,091102,,*01
$GPGGA,235950.000,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7E
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,235950.000,V,0000.0000,N,00000.0000,E,0.000000,,091102,,*09
$GPGGA,235951.000,0000.0000,N,00000.0000,E,0,00,50.0,0.0,M,0.0,M,0.0,0000*7F
$GPGSA,A,1,,,,,,,,,,,,,50.0,50.0,50.0*05
$GPRMC,235951.000,V,0000.0000,N,00000.0000,E,0.000000,,091102,,*08
…
```

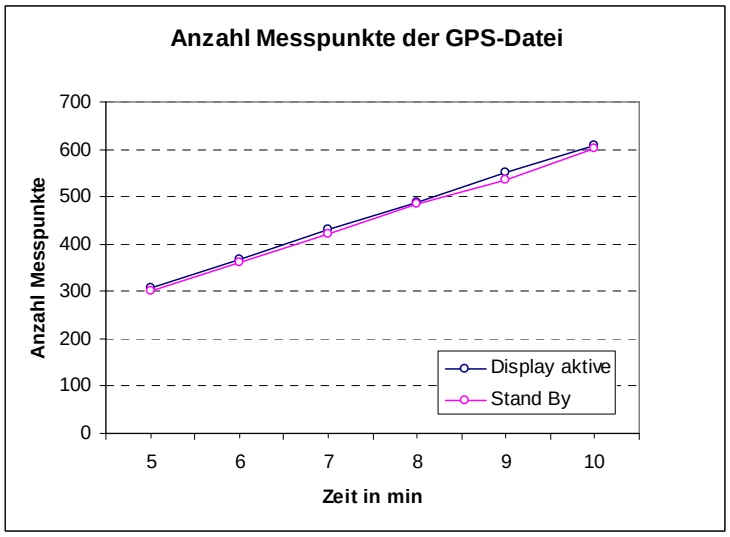
(Abb. 31: GPS-Daten)

Um auch für die GPS-Daten sicherzustellen, dass während dem „Standby-Modus“ aufgenommen wird, wurden wieder die Grösse der anfallenden Dateien während einer Messung im „Standby-Modus“ und einer Messung während dem das Display aktiv war, gemessen und verglichen.



(Abb. 32: Grösse der GPS-Dateien)

Die Ergebnisse dieser Messungen fallen für beide Fälle (Aktiv oder Standby) wieder fast gleich aus. Die Grösse der GPS-Datei, welche während dem „Stanby-Modus“ aufgenommen wurde, ist aber ganz minim kleiner. Dies lässt sich dadurch erklären, dass, während dem das Display nicht aktiv ist, das Telefon immer wieder versucht, in den „Standby-Modus“ zuwechseln und so zeitweise die Prozessorleistung zu reduziert wird. Dies gelingt aber nicht, da der Treo durch die Sensoraufnahmen immer wieder daran gehindert wird in den „Standby-Modus“ zu wechseln. Um diese Messungen zu konsolidieren wurden zusätzlich die Anzahl aufgenommener Messpunkte[7] gemessen. Aus dieser Messung kann, nebst der Kontrolle der ersten Messung, noch erkannt werden, dass die Datenstruktur regelmässig ist und immer gleichmässig viele Messpunkte aufgenommen werden.



(Abb. 33: Anzahl Messpunkte der GPS-Datei)

[7] Drei Zeilen entsprechen einem Messpunkt des GPS

4.2.4 Interne Variabeln

Das Programm wurde noch mit einer Funktion ergänzt, welche alle für uns wichtige interne Variabeln des Treo's ausliest und speichert. Da diese Funktion ebenfalls nicht in den Hintergund geschickt werden kann, wird diese Funktion auch wie die GPS-Funktion der Reihe nach, seriell abgearbeitet, sobald der Event Manager nichts zu tun hat (und einen `NILEvent` sendet).
Da ein zu häufiges Abfragen dieser internen Variabeln zu Fehlern führt und da diese Variabeln keinen starken Änderungen unterliegen[8], wurde ein Timer eingeführt, so dass diese internen Variabeln nur einmal pro Sekunde abgefragt werden. Folgende Variabeln werden abgefragt:

- Als erstes wird überprüft, ob die Tastensperre aktiviert oder deaktiviert ist. Dies kann darüber Auskunft geben wann der Benutzter das Telefon verwendet hat und wann es im „Standby-Modus" war.
- Danach wird die Signalstärke abgefragt. Falls kein Signal vorhanden ist wird der Signalstärkewert 99, ansonsten wird der Wert der Signalstärke in dBm[9] zurückgegeben.
- Als nächstes wird der Batteriestatus abgefragt. Diese Funktion gibt in Prozent die aktuelle Batteriestärke zurück.
- Und zu guter Letzt wird noch überprüft, ob das Telefon auf „mute" gestellt ist, oder ob der Lautsprecher aktiviert ist.

Um diese Variabeln abzufragen wurden folgende API-Aufrufe verwendet:

```
//Überprüfe ob Tastensperre aktiv war
HsAttrGet(hsAttrKeyboardLocked, 0, &keyguardActive);

//Signalstärke abfragen
PhnLibSignalQuality(PhoneLibRefNum, &quality);

//Batteriestärke abfragen
SysBatteryInfo (false,NULL,NULL,NULL,NULL,NULL,percentP);

//Überprüfen, ob Ton aktiviert ist
HsAttrGet(hsAttrRingSwitch, 0, &ringSwitch);
```

[8] D.h. innerhalb einer Sekunde

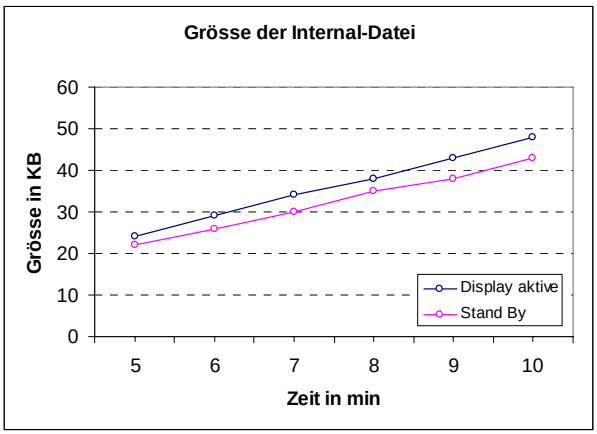
[9] **dBm** bezeichnet einen Signalpegel, bezogen auf die Leistung von 1 Milliwatt mW in logarithmischem Maßstab Wikipedia. (2006). from http://de.wikipedia.org/wiki/Hauptseite.

Die ausgelesen Variabeln, werden pro Durchgang zu einer Zeile zusammengefasst. Diese Daten sehen folgendermassen aus:

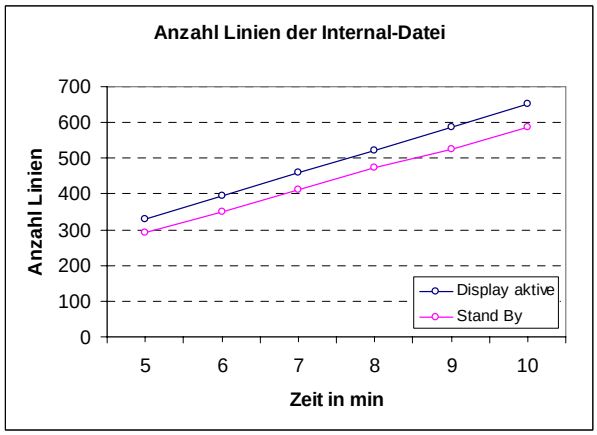
```
…
 keine tastensperre,signal strength: 21,battery strength: 5,silent
 keine tastensperre,signal strength: 21,battery strength: 5,silent
 keine tastensperre,signal strength: 21,battery strength: 5,silent
 keine tastensperre,signal strength: 21,battery strength: 5,silent
 keine tastensperre,signal strength: 21,battery strength: 5,silent
 tastensperre aktiv,signal strength: 21,battery strength: 5,silent
 tastensperre aktiv,signal strength: 21,battery strength: 5,silent
 tastensperre aktiv,signal strength: 20,battery strength: 5,silent
 tastensperre aktiv,signal strength: 20,battery strength: 5,silent
 tastensperre aktiv,signal strength: 20,battery strength: 5,silent
 tastensperre aktiv,signal strength: 20,battery strength: 5,silent
…
```

(Abb. 34: interne Variabeln)

Auch für diese Sensorgruppe wurde die Grösse und die Anzahl Messpunkte, über die Zeit hinweg, im „Standby-Modus“ und mit aktiviertem Display, gemessen und verglichen.



(Abb. 35: Grösse der Intern-Datei)



(Abb. 36: Anzahl Messpunkte der Intern-Datei)

Diese Messung bestätigt wieder das mit der GPS-Datei erhaltene Resultat. Es wird regelmässig aufgenommen, aber im „Standby-Modus“ wird etwas weniger abgespeichert. Hier wird der Unterschied zwischen „Standby-Modus“ und aktiviertem Display besser sichtbar als bei der letzten Messung. Die Abweichung variiert zwischen 8-10 %.

4.2.5 Photo

Die Funktion, welche in regelmässigen Abständen mit der integrierten Kamera des Treo's Bilder machen sollte, hatte bei meinem Vorgänger Robin Bucciarelli [Bucciarelli, 2006] nicht funktioniert. Da meine Prioritäten auf der Umsetzung des Ablaufes des Programms und des Speicherns der Sensordaten lag, habe ich mich nur am Rand mit der Photofunktion auseinandergesetzt.
Bei der Suche nach einem Beispielcode für die Photofunktion bin ich auf das Freeware Spiel „Video Jigsaw" [Ulatowski, 2006] gestossen. Da Szymon Ulatowski in diesem Spiel die Kamera des Treo's erfolgreich einsetzt, habe ich ihm ein Email geschickt und Ihn gefragt wie man vorzugehen hat, um im Hintergrund ein Photo zu schiessen und dies auf der SD-Karte abzuspeichern. Er hat mir sehr detailliert geantwortet und erklärt wie vorzugehen ist. Das Email mit der Antwort findet sich im Anhang (Anhang 9.5). Obwohl seine Anweisungen eins zu eins umgesetzt wurden, speichert das Programm jedoch immer noch keine Photos ab.
Um trotzdem den benötigten Speicherplatz für die Bilder berücksichtigen zu können, habe ich mit der Kamera Bilder gemacht und sie in verschieden Auflösungen und Komprimierungsstufen gespeichert.


(Abb. 37: Jpeg 640x480/ 26.6 KB)

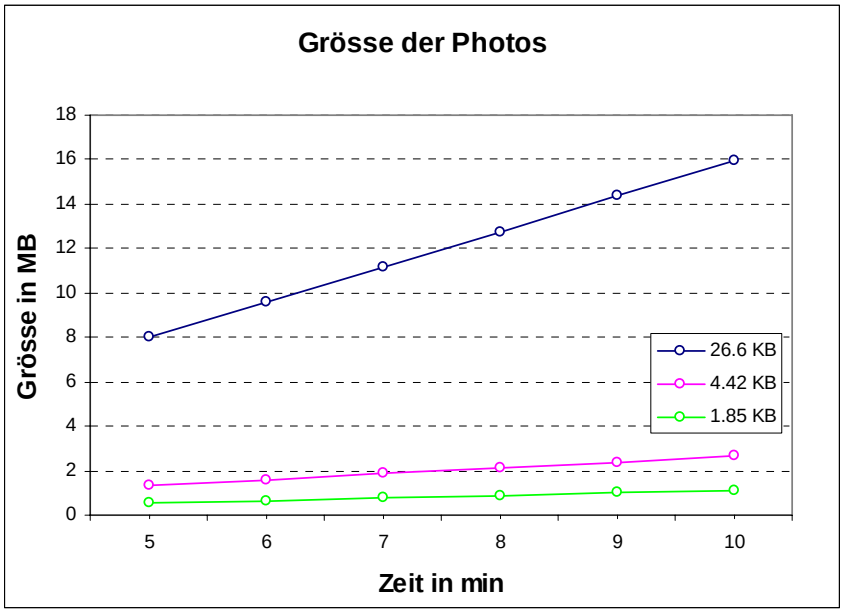

(Abb. 38: minimale Qualität 640x480/4.42 KB)

Die standartmässige Auflösung der Kamera des Treo's ist 640x480. Ein Photo wird im Jpeg-Format abgespeichert und benötigt 26.6 KB. Durch Erhöhung der Kompression bzw. Verminderung der Qualität wurde das Bild auf eine Grösse von 4.42 KB reduziert. Mit einer Verkleinerung der Auflösung auf 320x240 lässt sich das Bild sogar auf die Grösse von 1.85 KB verringern.[10]


(Abb. 39: Jpeg /minimale Qualität 320x240/ 1.85 KB)

[10] Die Komprimierung des Bildes wurde mit Adobe Photoshop vorgenommen

Um den benötigten Platzbedarf auf zu zeigen wurde im folgenden Diagramm, für die drei erwähnten Bildgrössen, der benötigte Speicherbedarf über die Zeit hinweg dargestellt.

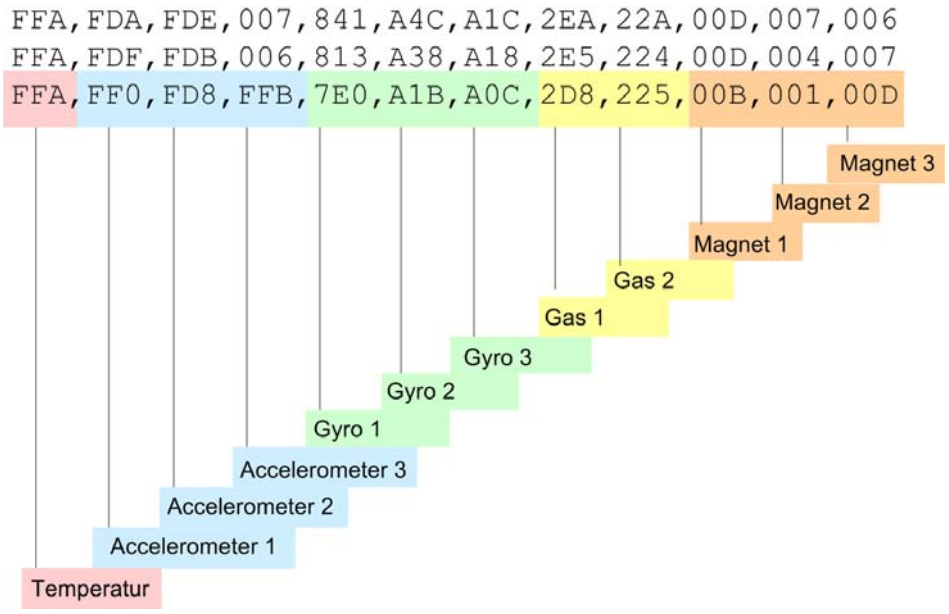


(Abb. 40: Speicherbedarf Photo)

4.2.6 Sensorboard / serielle Schnittstelle

Um die Daten des Sensorboardes empfangen und speichern zu können muss als erstes die serielle Schnittstelle des Treo's angesprochen und geöffnet werden. Ist diese bereit können die Sensorwerte des Boards über das Serial-Protokoll empfangen werden. Die serielle Schnittstelle und das entsprechende Protokoll ist bereits in Robin Bucciarelli's Diplomarbeit [Bucciarelli, 2006] beschrieben, daher wird hier nicht mehr darauf eingegangen.

Die Daten, welche über die serielle Schnittstelle von dem Senosorboard empfangen werden, sollten folgendermassen aussehen:



(Abb. 41: Daten des Sensorboardes)

Es sind zwölf hexadezimale Dreiergruppen, welche jeweils den Wert eines Sensors wiedergeben. Somit entspricht eine Zeile einer Messung aller Sensoren. Die nächste Messreihe folgt auf einer neuen Zeile.

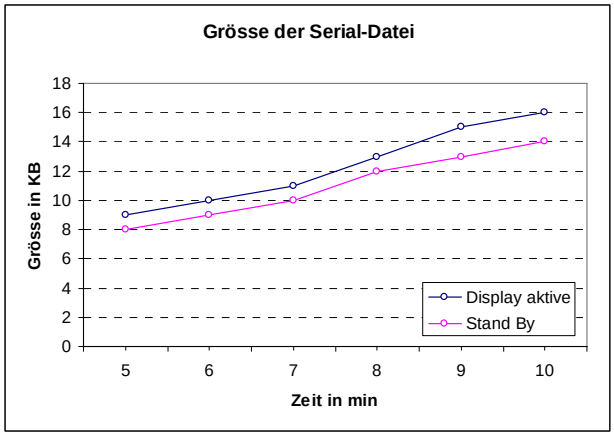
Diese Funktion laufft in Robins Programm nur bedingt. Die Daten werden nur teilweise und in sehr kleinen Fragmenten empfangen, so dass die erhaltnen Daten völlig zerstückelt sind und nicht verwendet werden können.

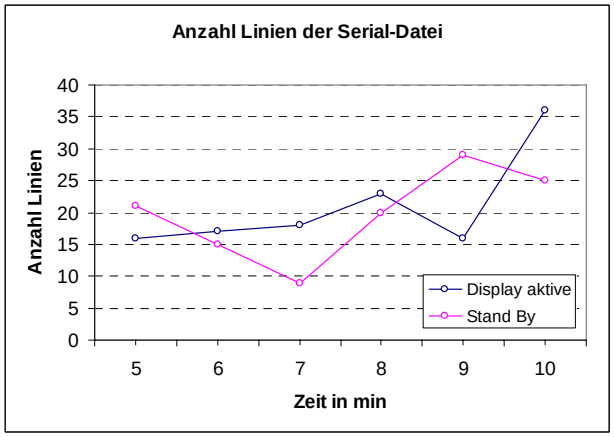
Die gespeicherten Daten von Robins Programm sehen im Vergleich folgendermassen aus:

```
_Yhh¡,339,228,00B,FFB,00CC63,B8E,A84,330,230,008,FFB,003FCC,008,C50,B86,A80,3
28,238,FFB,00C,00C4,A7F,31E,231,FFB,008,002FDE,FD7,FFB,C3B,B82,A81,330,232,00
9,FFB,0088,B7D,A84,32F,235,00D,007,006FFA,FD1,FDB,00B,C2F,B7B,A7E,32E,238,008
,006,004D,C2E,B7B,A7E,330,239,00A,002,00E32A,236,008,007,000B,00C,C1F,B75,A74
,329,229,005,005,005A79,32A,230,009,FFB,0071,FD1,000,C14,B75,A76,324,22D,FFB,
003,007A,FD0,FD1,004,C04,B6B,A7C,32B,229,007,006,007BFD,B71,A7A,332,235,00C,0
05,004

FFA,FD1,FDB,00D,BFE,B71,A7A,331,239,00A,FFB,00BFFB,BF3,B6D,A7A,32F,233,009,00
D,0076,327,22F,007,006,004FC6,FFB,BDF,B6C,A78,32A,22F,001,0
…
```

(Abb. 42: Sensorboarddaten Robin)

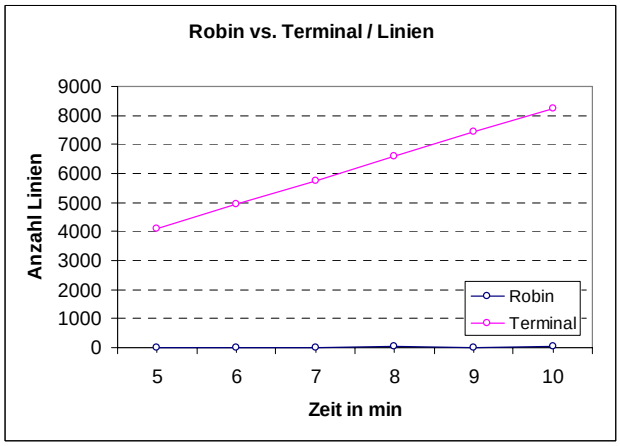
Dieser Sachverhalt widerspiegelt sich ebenfalls in einer Messung, in welcher analog zu den vorherigen Sensorgruppen die aufgenommene Grösse und die Anzahl der Zeilen gemessen wurden.

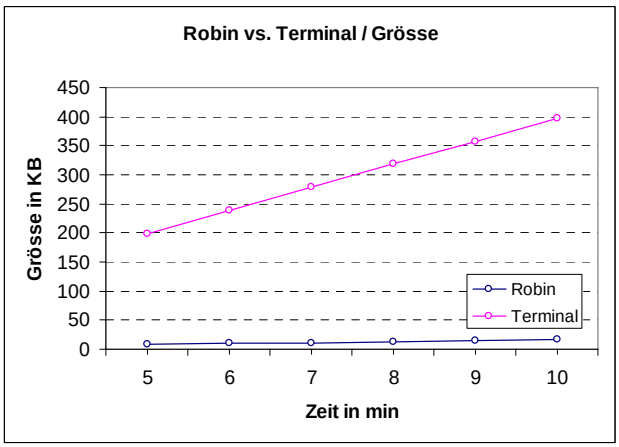




(Abb. 43: Grösse der Serial-Datei) (Abb. 44: Anzahl Linien der Serial-Datei)

Die Grösse der Datei nimmt zwar gleichmässig zu, aber die Anzahl der Linien ist bei den einzelnen Messungen willkürlich verschieden gross. Weiter fällt auf, dass in 10 Minuten Aufnahmezeit lediglich zwischen 25 und 35 Linien aufgenommen werden.
Mein Verdacht, dass Robins Programm nicht nur die Daten durcheinander bringt, sondern, dass es auch massiv zu wenig aufnimmt, erhärtet sich durch folgende Messung. Es wurde die Grösse und die Anzahl Zeilen zusätzlich mit dem Terminalprogramm CS Online [Conklinsystems, 2006], gemessen und diese Ergebnisse mit denen des Programms von Robin verglichen.

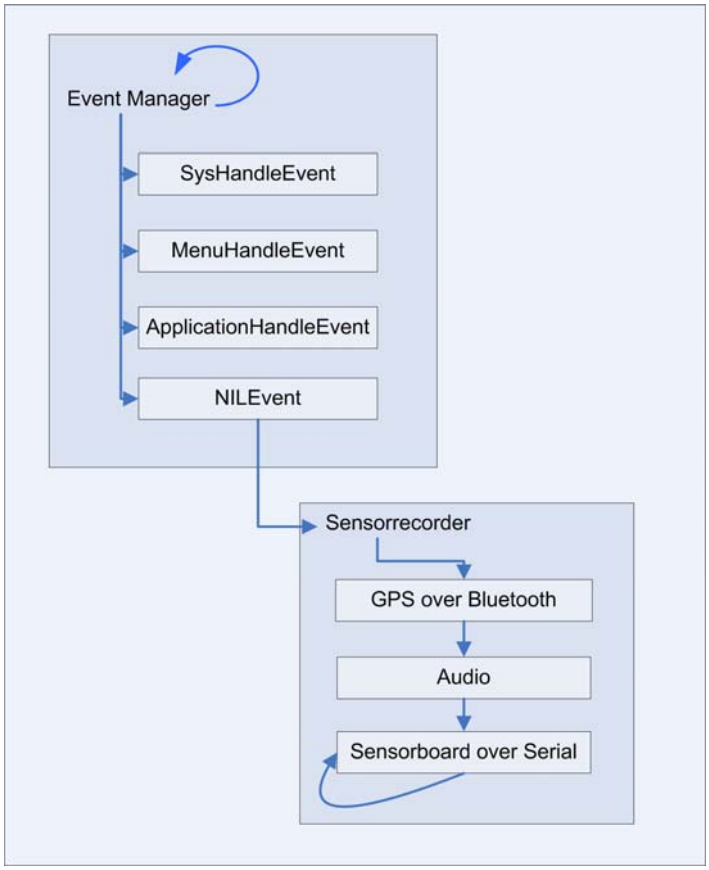




(Abb. 45: Grösse der Serial-Datei) (Abb. 46: Anzahl Linien der Serial-Datei)

Aus dieser Messung geht hervor, dass Robins Programm nur ca. 1/40stel der vorhandenen Daten aufnimmt.

Dieses Verhalten entstammt aus einer Fehlüberlegung beim Design des Programms. Robins Programm ist folgendermassen aufgebaut: Sobald der Event Manager nichts zu tun hat und einen `NILEvent` sendet, werden als erstes drei zusammengehörende Zeilen der GPS-Daten empfangen. Ist dies getan wird überprüft, ob für den „Audiostream“ eine Zustandsänderung, wie den Stream anhalten, starten oder Pause, nötig ist. Sind diese Pendenzen abgehandelt, wird über die bereits beim Starten des Programms geöffnete Verbindung eine Funktion aufgerufen, welche ein Fenster öffnet und die einströmenden Daten, über die serielle Schnittstelle, in einen Buffer umleitet. Sobald dieser Buffer voll ist, wird dieses Fenster wieder geschlossen. Dies würde wunderbar funktionieren, wenn das Programm nichts anderes als die Aufnahme der seriellen Daten zu tun hätte und die Funktion innerhalb des Event Manager ausgeführt und nicht durch einen `NILEvent` gestartet würde.



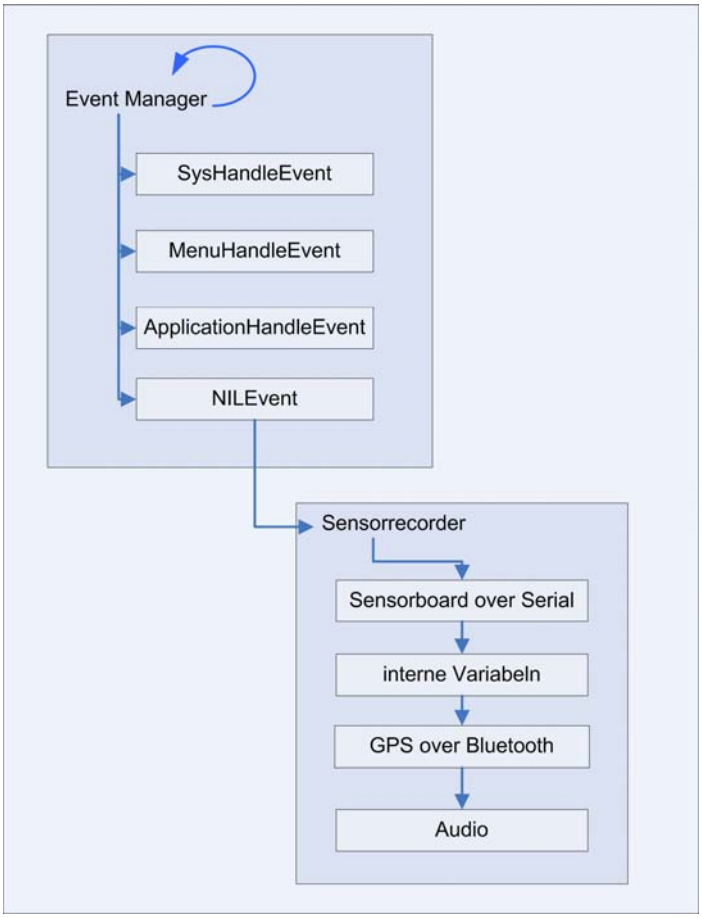
(Abb. 47: Ablauf Sensoraufnahme Robin)

Da der Event Manager aber innerhalb eines Systemticks[11] überprüft, ob ein Event abgehandelt werden muss und, falls nicht, wieder einen `NILEvent` startet. Dies bedeutet, dass die Funktion, welche das Fenster zum Aufnehmen der Sensordaten öffnet, jede 1/100stel Sekunde wieder von neuem gestartet wird und wieder ein Fenster öffnet. Die Funktion hätte aber wesentlich länger um den Buffer zu füllen (welcher Standardmässig auf 1024 Bytes fixiert ist) und anschliessend das Fenster wieder zu schliessen. So kann die Funktion das Fenster nicht schliessen und die Daten im Buffer werden von immer neu geöffneten Fenstern überschrieben. Wohl könnte man im Event Manager ein Timeout setzten, um der Funktion mehr Zeit zu geben das Fenster wieder zu schliessen, doch dieses Timeout entspricht eins zu eins der Reaktionszeit des Programms. Würde man dem Programm 200 Systemticks Zeit geben (was immer noch nicht ausreichen würde), so hätte

[11] Ein Systemtick unter PalmOS 5.2 Garnet entspricht 1/100erstel Sekunde

das Programm 2 Sekunden bis es auf irgendeinen Event reagiert. Das heisst wenn eine Taste gedrückt wird, ginge es mindestens 2 Sekunden bevor etwas passieren würde. Als Alternative wäre es möglich den Buffer der Aufnahmefunktion zu verkleinern. Doch dieser müsste so klein gewählt werden, dass in ihm lediglich ein bis zwei Zeilen der Sensorboardwerte platz hätten, um rechtzeitig wieder das Fenster schliessen zu können. Da aber beim Öffnen des Fensters nicht unbedingt gerade der Anfang einer Zeile gesendet wird und ein beliebiges Teilstück der Zeilen aufgenommen wird, erhält man so auch nicht das gewünschte Resultat.

Um dieses Problem zu umgehen, wurde folgender Ansatz verwendet: Genau wie in Robins Programm wurde der `NILEvent` des Event Managers ausgenützt um die Funktionen zur Aufnahme der Sensoren zu starten. Nebst dem, dass neu eine Funktion zur Erfassung der internen Variabeln hinzugekommen ist, wurde die Reihenfolge so umgestellt, dass als erstes die Daten des Sensorboardes über die serielle Schnittstelle, dann die internen Variabeln, anschliessend die Daten des GPS aufgenommen werden und als letztes die Zustandsänderungen des „Audiostreams“ angepasst werden.



(Abb. 48: Ablauf Sensoraufnahme neu)

Diese Umstellung hat zur Folge, dass die Funktion zur Speicherung der Daten des Sensorboardes als erstes gestartet wird und somit Priorität über die anderen hat. Zusätzlich wurde sie so verändert, dass nicht mehr ein Fenster geöffnet und ein Buffer aufgefüllt wird.[12]

[12] Für PalmOS gibt es zwei verschiedene Funktionen um Daten über die serielle Schnittstelle zu empfangen. Die Erste ist die beschriebene Methode mit dem Fenster, die Zweite greift direkt aus dem Buffer eine bestimmte Menge von Bytes ab.

Um den neuen Ansatz realisieren zu können wurde auf den Sensorboards zwischen jeder Zeile eine Pause (Zeit in der keine Daten gesendet werden) eingefügt. Der Ablauf der Funktion sieht nun folgendermassen aus. Als erstes wird der Buffer des Treo internen „Serialprotokolls“ gelehrt und solange gewartet, bis eine bestimmte Zeit keine Daten kommen (Dies entspricht der Pause des Sensorboardes). Sobald diese Pause erkannt wird, wird die Aufnahme gestartet. Es wird aber nur eine festgelegte Anzahl von Bytes abgegriffen. Durch die Pause weiss man, dass, sobald die Aufnahme beginnt der Anfang einer Zeile kommt und mit der gewählten Anzahl Bytes kann man genau eine oder zwei Zeilen abgreifen. Ist dies erledigt, werden die anderen Funktionen (interne Variabeln, GPS & Audio) abgehandelt.

Die API-Aufrufe für diese Funktion sehen folgendermassen aus:

```
// Buffer Leeren… bis „time” –lange keine neue Daten kommen
SrmReceiveFlush(gPortId,time);

//Warten bis n Bytes im Buffer sind oder die Zeit „waitTime“
//vorbei ist
SrmReceiveWait(gPortId,Bytes,waitTime);

//Die sich im Buffer befinden Daten abgreifen -> theDate ist
//Pointer auf die empfangenen Daten
SrmReceive(gPortId, &theData, numBytesPending, 0,&err);
```

Mit diesem Ansatz ist es etwa möglich 20 Zeilen pro Sekunde abzugreifen. Dies sollte für die Requirements ausreichen. So ist es möglich mit 20 Messpunkten pro Sekunde mit dem Accelerometer einen einzelnen Schritt aufzulösen. Ein erster Versuch hat folgendes Resultat geliefert:

```
…
FFA,FE1,FEA,00A,7C0,A14,A0B,2C4,22E,00C,006,005
00A,7C4,A0E,A09,2C2,223,007,FFB,004
00A,7C4,A0E,A09,2C2,223,007,FFB,004
FFA,FC4,F5A,003,7C7,A42,A01,2C3,21F,FFB,FFB,005
7,A09,A06,2C1,228,009,FFB,001
FFA,FE1,FEA,00B,7C0,A14,A0B,2C4,22E,00B,006,004
A,FE2,FE1,002,7C2,A0B,A0F,2BA,231,FFB,002,005
4,007
A,FFB
F06,FCA,005,7C7,720,A17,2B5,233,000,006,00B
FE7,003,7BE,A0D,A03,2C1,228,FFB,00D,00C
FE7,003,7BE,A0D,A03,2C1,228,FFB,00D,00C
A,008,7C5,A10,A09,2BC,231,006,00A,005
5,007
FFA,FEC,FDA,008,7C5,A12,A0C,2B9,22B,009,006,005
5,7C7,A0E,A04,2BE,22A,00A,009,007
5,7C7,A0E,A04,2BE,22A,00A,009,007
FFA,FE1,FEA,00B,7BC,A0B,A0E,2BE,230,003,006,009
…
```

(Abb. 49: Sensorboarddaten neu)

Hier werden schon einige Zeilen richtig empfangen und haben die gewünschte Form von zwölf mal drei Zeichen mit einem Komma getrennt. Bei den Restlichen wurde der Anfang der Zeile nicht aufgenommen.

Dies lässt sich sehr wahrscheinlich beheben, indem man die richtige Einstellung für die Parameter wählt. Die 6 Parameter, welche variiert werden können sind:

- Die Baudrate des Sensorboards kann verändert werden. So werden die Sensorwerte verschieden schnell gesendet. (Für die vorherige Messung war das Board auf 38'400 Baud eingestellt)
- Die Pausenlänge des Sensorboards zwischen den Zeilen kann variiert werden. (Die Pause betrug 1,5 Hundertstelsekunden
- Das „Timeout“ des Event Managers kann verändert werden (Hier war er auf 1/100stel Sekunden eingestellt)
- In der Funktion, in welcher der Buffer geleert wird, kann die Zeit, wie lang keine Daten kommen müssen, damit die Funktion beendet wird, eingestellt werden. (Sie war auf 1/100stel Sekunde eingestellt.)
- In der Funktion, welche wartet, bis eine gewisse Menge sich neu im Buffer befindet, kann die Anzahl Bytes, auf welche gewartet wird, angepasst werden. (Hier 50 Bytes, dies entspricht einer Zeile)
- Zusätzlich kann in dieser Funktion noch ein „Timeout“ eingestellt werden, damit falls keine Daten kommen, die Funktion weiter geht. (1/100 Sekunde)

Bei dem Einstellen der Parameter muss darauf geachtet werden, dass das „Timeout“ des Event Managers gleich oder grösser ist als das „Timeout“ der Wartefunktion. Falls dies nicht so ist, kann, wenn das Programm geschlossen wird, evtl. der serielle Port geschlossen werden, solange die Funktion noch wartet. Versucht diese Funktion nun die Daten des geschlossenen Ports zu empfangen führt dies zu einem Absturz des Telefons. So muss nebst den erwähnten Parameter stehts auch die Robustheit des gesamten Programms wieder überprüft werden.

Von Zeit zu Zeit müssen die Magnetsensoren des Sensorboards geresetet werden. Da diese Sensoren mit der Zeit selbst depolarisiert werden und so verfälschte Werte liefern. Das Reseten der Magnetsensoren wird auf dem Sensorboard gelöst, indem der Sensor einem starken internen Magnetfeld (durch einen hohen Stromstoss in der ummantelnde Spule) aus gesetzt wird. Dies hat zur folge, dass die Magnetsensoren wieder polarisiert sind. [Honeywell, 2006]
Dieser Prozess wird gestartet, sobald über die serielle Schnittstelle kurz nacheinander ein kleines und ein grosses „F“ geschickt werden. Für diesen Zweck ist ein Timer eingeführt worden, welcher alle 2 Minuten die Magnetfeldsensoren neu kalibriert.

Der Befehl, um Daten über die serielle Schnittstelle abzuschicken sieht folgendermassen aus:

```
//Daten über serielle Schnitstelle senden

//gPortID = Adresse der geöffneten Schnitstelle
//message = zu sendende Nachricht
//numBytesToSend    =    Anzahl    Bytes    der    Nachricht;
//err = Fehlermessage

SrmSend( gPortId, message, numBytesToSend, &err );
```

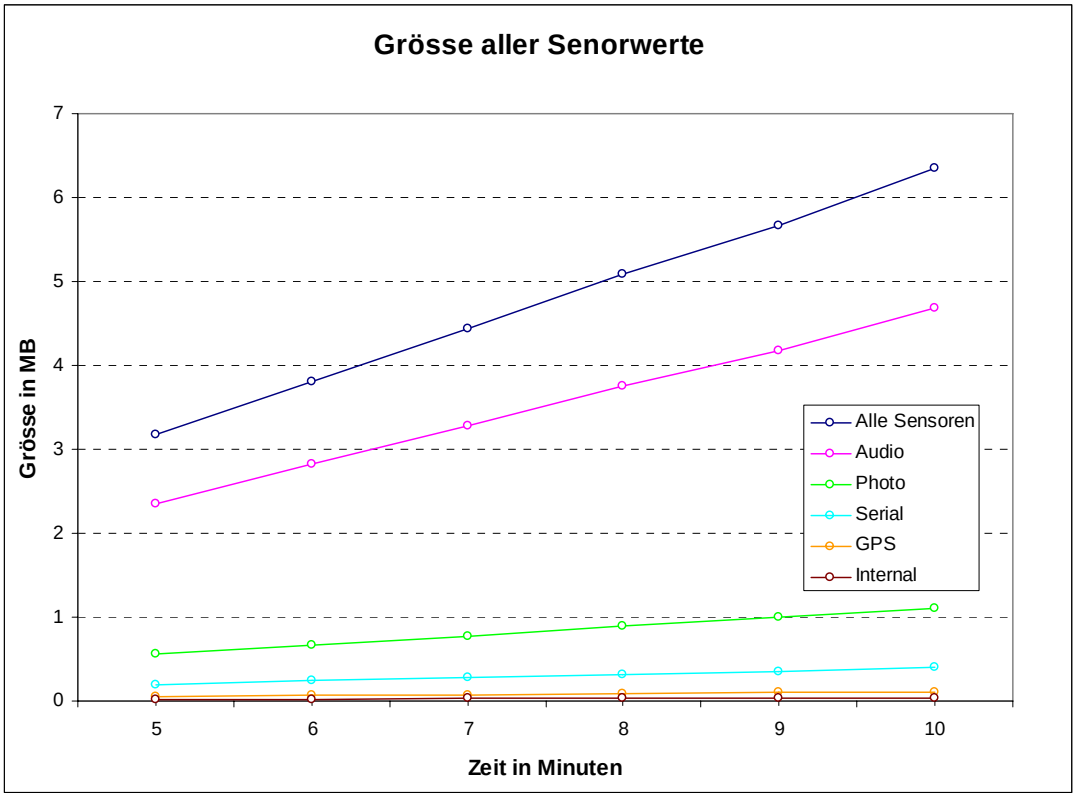
Zusätzlich kann durch das Senden von Buchstaben alle Sensorgruppen des Boardes einzeln ein- oder ausgeschaltet werden. In der folgenden Tabelle sind alle Befehle und ihre Bedeutung aufgelistet:

Temperatursensor	T = einschalten; t = ausschalten
Accelerometer	A = einschalten; a = ausschalten
Gyrosensor	G = einschalten; g = ausschalten
Gassensor 1	X = einschalten; x = ausschalten
Gassensor 2	Y = einschalten; y = ausschalten
Gassensor 3	Z = einschalten; z = ausschalten
Magnetfeldsensoren	M = einschalten; m = ausschalten
Reset der Magnetfeldsensoren	„F“ gefolgt von „f“

(Abb. 50: Befehle für Sensorboard)

Die Sensoren sind immer standardmässig eingestellt, bis auf die Gassensoren. Wegen ihres sehr hohen Energieverbrauchs, können die Gassensoren nur sporadisch aktiviert werden. Wir haben uns dafür entschieden, dass, sobald ein eingehender Anruf beendet wird, die Gassensoren während dem Ausfüllen des Menus, welches im Kapitel 4.1.3 beschrieben ist, gestartet werden und ca. 40 Sekunden aufgenommen werden. [13]

Als Abschluss dieses Kapitels wurde im folgenden Diagramm noch von allen Sensorgruppen die benötigte Speicherkapazitäten und deren Summe dargestellt um einen Überblick über den benötigten Speicherplatz zu erhalten.



(Abb. 51: Grösse aller Sensordateien)

[13] Die Gassensoren benötigen ungefähr 20-30 Sekunden, bis sie aufgewärmt sind und die korrekten Werte anzeigen

4.3 Abspeicherung der Daten

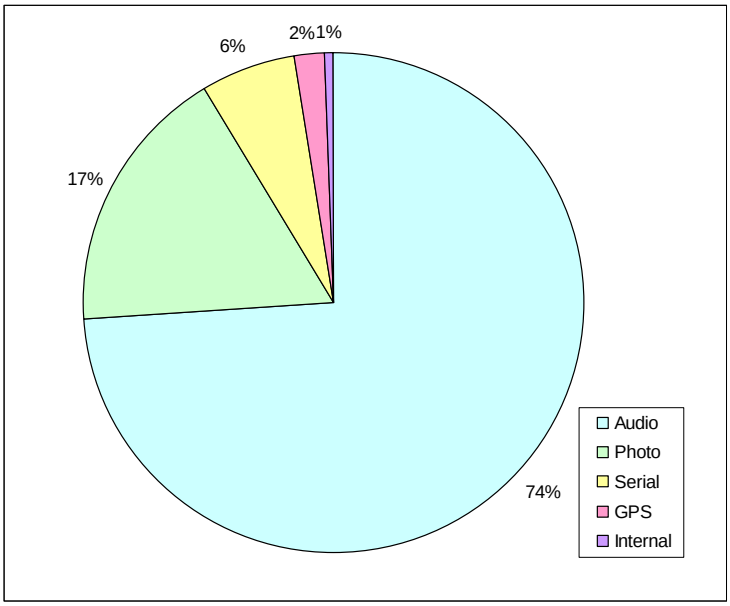
Die Daten, welche durch die internen und externen Sensoren des Telefons aufgenommen wurden, müssen nun in einer wieder verwendbaren Form gespeichert werden. Da eine relativ grosse Datenmenge durch all die Sensoren zusammen kommt, ist es nahe liegend, die Daten auf eine SD-Karte zu speichern. Hierzu werden SD-Karten verwendet, welche eine Speicherkapazität von 2GB umfassen. Dies ist, wie in der Beschreibung des Treo's bereits erwähnt, die maximale Grösse für SD-Karten, welche vom Treo erkannt werden. [14]

4.3.1 Speicherbedarf

Würde man nun alle Sensorwerte immer aufnehmen, so werden in 10 Minuten ca. 6.5 MB Daten erzeugt (siehe Abb. 51). Diese 6.5 MB kommen folgendermassen zusammen:

- Eine 10-minütige Audioaufnahme in der tiefsten, auf dem Treo verfügbaren, Speicherqualität (Mono 8 KByte /s) ergibt 4,8 MB.[15]
- Die Abfrage der internen Sensoren produziert in einer Sekunde etwa eine Linie mit 70 Zeichen. Dies ergibt ca. 48 KB für 10 Minuten.
- Der GPS Empfänger generiert ca. 190 Bytes pro Sekunde. So kommen in 10 Minuten etwa 110 KB.
- Über die serielle Schnittstelle kommt durch das Sensorboard ca. 10 Zeilen a 50 Zeichen pro Sekunde, was für die 10Minuten 450 KB ergibt.
- Da zu diesem Zeitpunkt die Integration des Photomoduls noch nicht abgeschlossen ist, wird der im Kapitel 4.2.5 ermittelte Wert von 1.85 KB pro Bild angenommen. Das ergibt für ein Bild pro Sekunde in 10 Minuten 1,1 MB

Diese Grössen aufaddiert, ergibt 6.5 MB. In dem folgenden Diagramm lässt sich die prozentuale Verteilung des Platzbedarfs der einzelnen Sensorgruppen erkennen.

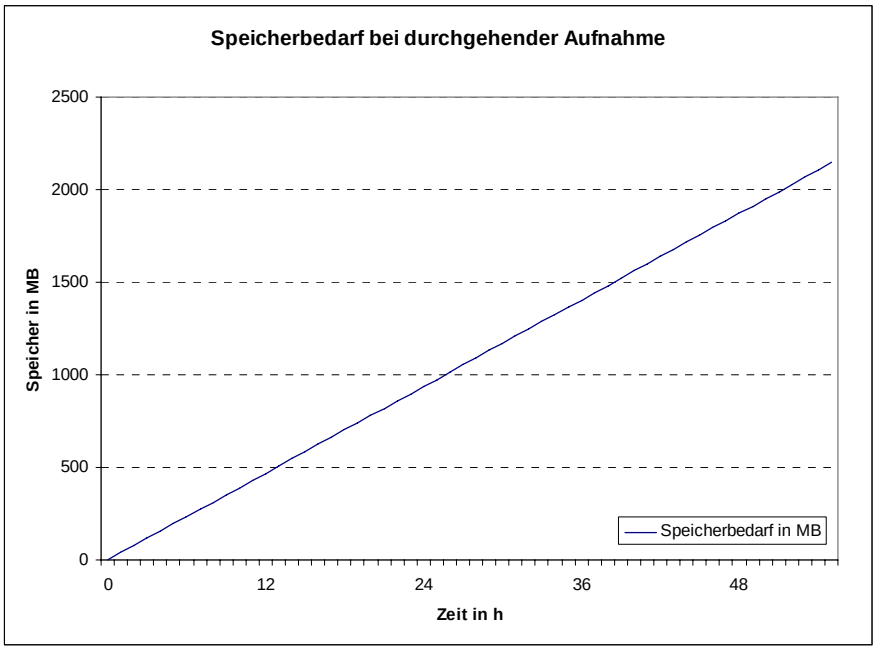


(Abb. 52: Verhältnis der Grössen der Sensordaten)

[14] 2GB SD-Karten waren die grössten verfügbaren Karten zu dem Zeitpunkt wo dies getestet wurde. Mittlerweile werden schon 4GB SD-Karten angeboten. Ob 4GB Katen auf dem Treo 650 erkannt werden müsste noch getestet werden

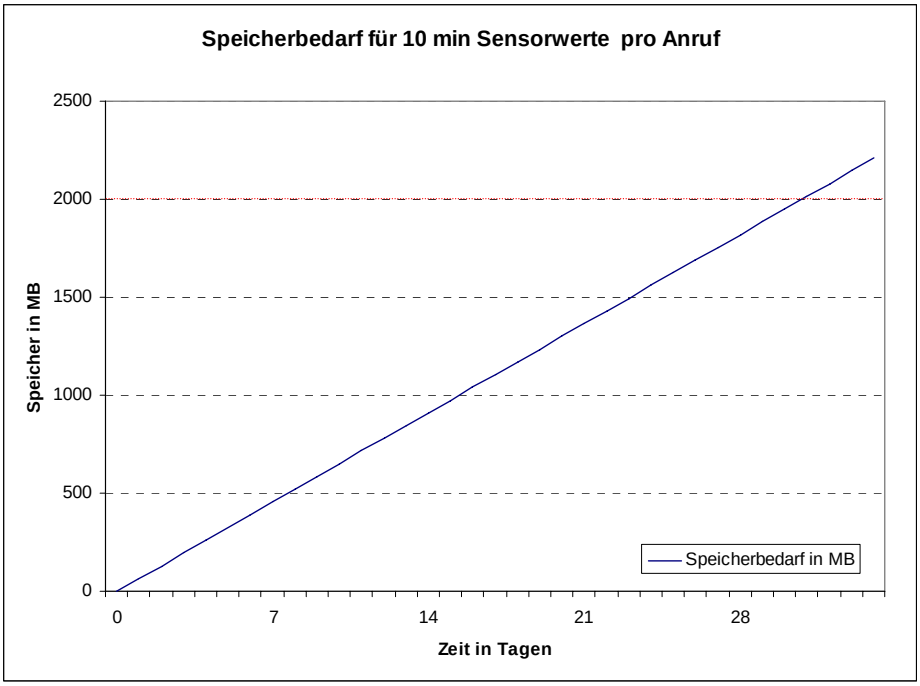
[15] Wie in Kapitel 4.2.2 beschrieben, kann der Speicherbedarf für Audio halbiert werden, durch Konvertierung zu MP3 mit 32Kbit/s

Diese würde bedeuten, dass für eine Daueraufnahme die 2 GB-Karte innerhalb von etwa 56 Stunden voll wäre und so mindestens alle zwei Tage die Karte gewechselt oder gelöscht werden müsste. Da das Experiment über längere Zeit (zwei bis drei Monate) hinweg geplant ist, ist es wichtig, dass die Handhabung für den Benutzer so einfach wie möglich ist und sich der Benutzer so wenig wie möglich um Dinge, wie das Sichern und Leeren der SD-Karten, kümmern muss, um so nicht die Benutzerakzeptanz einzuschränken, indem spätestens an jedem zweiten Tag die SD-Karte über einen „Card-Reader“ gesichert werden müsste.



(Abb. 53: Speicherbedarf bei durchgehender Sensoraufnahme)

Hier hinzu kommt auch noch die Tatsache, dass für die Auswertung des Versuches gar nicht so viele Daten benötigt werden. Es genügt vollkommen, wenn man die Sensordaten von etwa 10 Minuten vor dem eingehenden Telefonat hat, um auf die Unterbrechbarkeit Rückschlüsse machen zu können. Gemäss des Versuchs mit dem „Sensoroffice“ reichen sogar schon 20 Sekunden [Vorburger 2, 2005]. Um aber auf der sicheren Seite zu sein und dieses Resultat bestätigen zu können sind 10 Minuten Aufnahmezeit wünschenswert.
Da aber nicht bekannt ist, wann der nächste Anruf eingeht, müssen die letzten 10 Minuten immer im Speicher gehalten und sobald ein Anruf eintrifft, aus dem Speicher auf die SD-Karte gespeichert werden. Rechnet man nun damit, dass bei einem Probanden ca. 10 Anrufe pro Tag eingehen so entstehen pro Tag ungefähr 60 MB Daten. So ist die SD-Karte erst in etwa einem Monat voll und muss während dem Experiment nur 2- bis 3-mal gesichert und gelöscht werden. So kann einmal pro Monat einfach die SD-Karte ausgetauscht werden und der Proband muss nicht zusätzlich einen „Card-Reader“ verwenden.

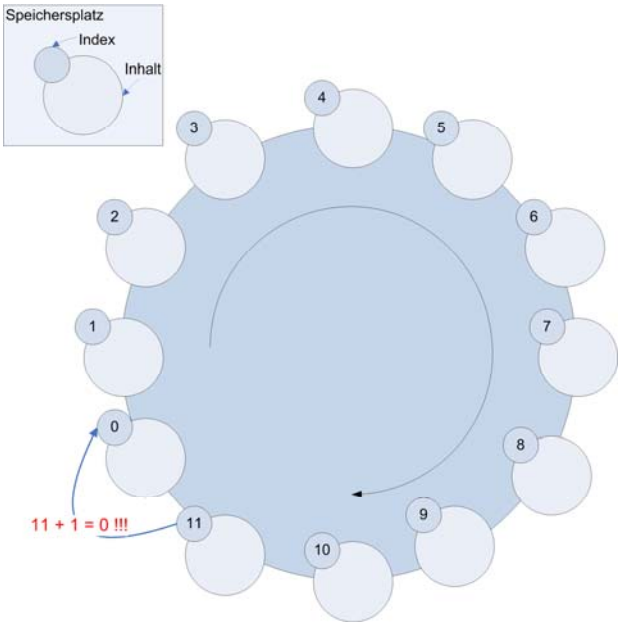


(Abb. 54: Speicherbedarf für 10 min Sensoraufnahme pro Anruf)

Um es zu ermöglichen, dass sich immer die letzten 10 Minuten im Speicher befinden, muss ein FIFO-Buffer (First IN First Out) geschaffen werden, so dass, sobald ein neuer Sensorwert in den Buffer aufgenommen wird, der älteste Sensorwert gelöscht wird. Dies ist am einfachsten mit einem Ringbuffer umzusetzen. Bevor auf die konkrete Umsetzung des Ringbuffers in dem Programm eingegangen wird, möchte ich kurz grundlegend die Funktionsweise eines Ringbuffers erläutern.

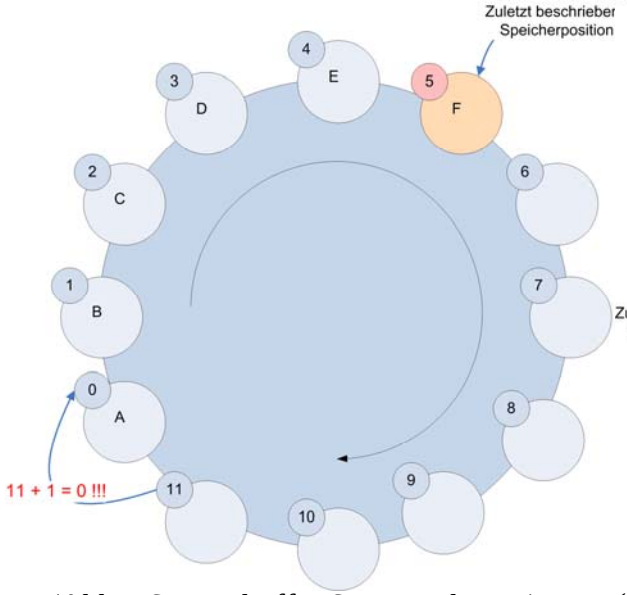
4.3.2 Ringbuffer

Um ein Ringbuffer zu erstellen, muss als erstes ein Buffer erstellt werden, in welchem genau so viele Speicherplätze alloziert sind wie benötig werden und diese Speicherplätze müssen über einen Index abrufbar sein. Die Idee des Ringbuffers ist nun, dass der Buffer normal von vorne nach hinten aufgefüllt wird. Sobald der letzte Speicherplatz erreicht wird, beginnt der Ringbuffer wieder beim ersten Speicherplatz und läuft so immer im Kreis.

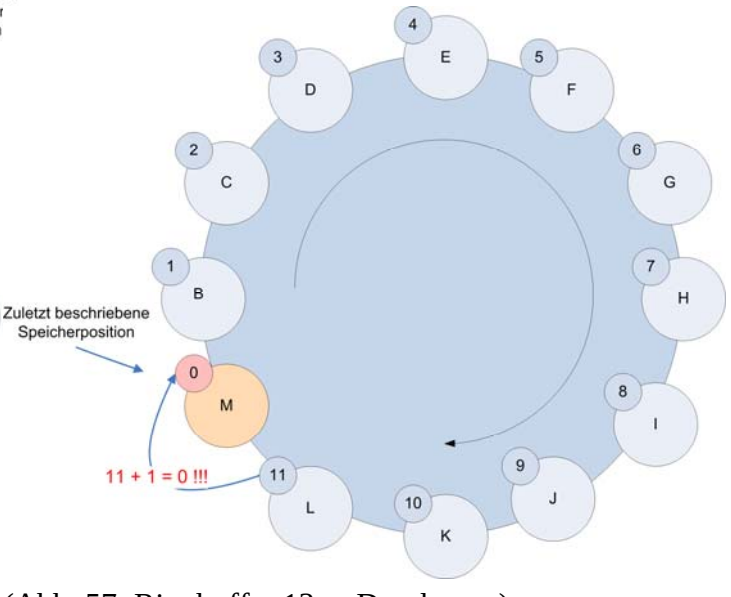


(Abb. 55: Ringbuffer)

Die genaue Funktionsweise wird am folgenden Beispiel erklärt: Wir nehmen einen Ringbuffer mit zwölf Speicherplätzen (0-11). In diesen Ringbuffer werden nun die zu speichernden Elemente abgefüllt. Hier wird für diesen Zweck das Alphabet verwendet. Ein Buchstabe entspricht einem zu speicherndem Element. Als erstes wird nun in die Speicherposition 0 ein A geschrieben, auf Position 1 ein B usw. Nach dem 6ten Durchgang sind die Speicherplätze 0-6 belegt und im 6-ten Platz ist ein F gespeichert (Abb. 56). Nach dem 12-ten Durchgang ist Speicherplatz 11 mit einem L belegt. Im 13-ten Durchgang wird nun wieder die Speicherposition 0 belegt (Abb. 57).

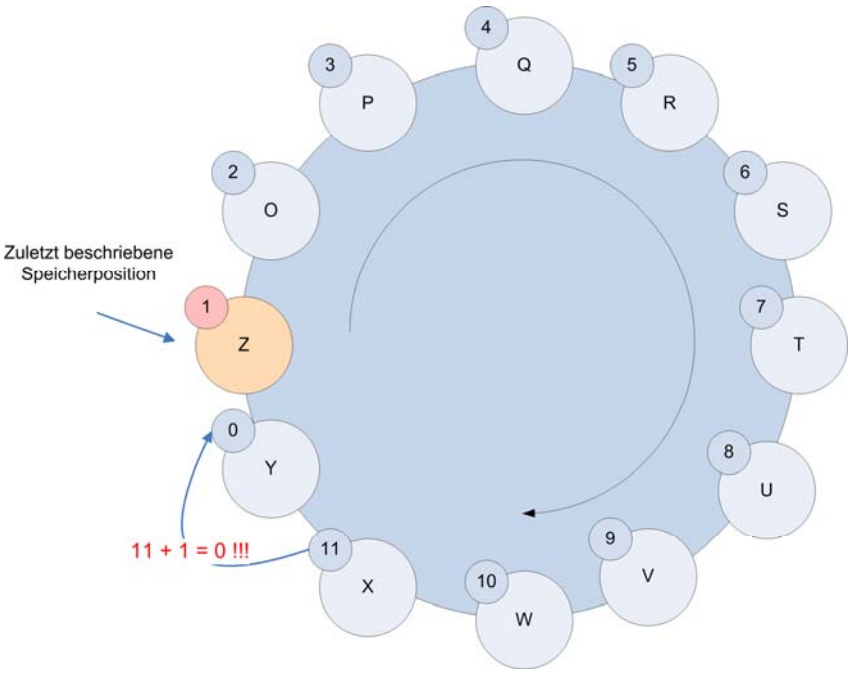


(Abb. 56: Ringbuffer 6ter Durchgang)



(Abb. 57: Ringbuffer 13ter Durchgang)

So läuft der Ringbuffer nun im Kreis, bis er unterbrochen wird oder bis alle Elemente abgefüllt sind. Für das Beispiel mit dem Alphabet ist der Zustand, wenn alle Buchstaben durchgegangen sind in folgender Graphik dargestellt.



(Abb. 58: Ringbuffer am Schluss)

Nun muss aber noch beachtet werden, dass dieser Buffer wieder in der richtigen Reihenfolge ausgelesen wird. Man beginnt mit dem Index der letzten beschrieben Position plus eins (2) und liest nun einen Speicherplatz nach dem anderen aus, bis zum höchsten Speicherplatz (11). Nachdem beginnt man beim ersten Speicherplatz (0) und liest noch alle Positionen bis und mit der letzten beschrieben Position(1) aus. Das so erhaltene Resultat ist nun: O,P,Q,R,S,T,U,V,W,X,Y,Z, was genau den letzten 12 Buchstaben des Alphabets entspricht.

4.3.2.1 Implementierung des Ringbuffer mittels Datenbank

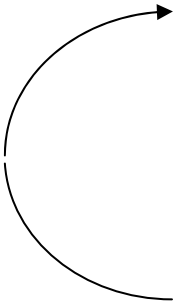
Wie im Kapitel 4.3.1 gezeigt, brauchen wir für den Ringbuffer für 10 Minuten Sensorwerte etwa 6 MB Arbeitsspeicher. Doch während dem Versuch einen Ringbuffer zu implementieren stellte sich heraus, dass der für ein Programm verfügbare Arbeitsspeicher bei weitem nicht für ein Ringbuffer dieser Grösse ausreicht. Selbst wenn man das Audio weg lassen würde sind die verbleibenden 512 KB immer noch zu gross für den verfügbaren Arbeitsspeicher, da wie im Kapitel 3.2.5 bereits erwähnt, der verfügbaren Speicher für ein Programm auf 68 KB begrenzt ist um Abwärtskompatibilität zu älteren Palms zu gewährleisten. Programmiert man einen Buffer, welcher grösser als 68 KB ist, wird das Programm vom Compiler nicht mehr kompiliert.
Es gibt zwar die Möglichkeit, selbst Speicher mit so genanten „Memory Chunks“ zu allozieren, welche ebenfalls eine maximale Grösse von 68 KB haben können. In diese Chunks könnte man nun Teile des Ringbuffers auslagern. Da man aber so beliebig Speicher be- oder überschreibt ist diese Methode unsauber und birgt das Risiko, dass Speicherkonflikte entstehen, welche wiederum zum Abstürzen des Telefons führen würden.
Da der Treo eher für ein Zielpublikum entworfen ist, welches den Treo geschäftlich einsetzt, muss er sehr viele Adressen, Notizen und Termin verwalten können. Für diesen

Zweck ist im Treo ein eigenes Datenbanksystem integriert. Auf der Suche nach einer geeigneten Methode den Ringbuffer umzusetzen ist mir die Idee gekommen, dieses Datenbanksystem für den Ringbuffer Zweck zu entfremden. Dies eignet sich sehr gut, da bereits 10 MB des Arbeitsspeichers als „DBChache“ reserviert sind und da jeder Datenbankeintrag mit einem Index versehen ist, mit welchem man diesen Eintrag wieder auslesen oder überschreiben kann. Auch die Transaktionsgeschwindigkeit ist genügend schnell, da die Transaktionen mit dem „DBCache“ den gleichen Speichertyp, wie der restliche Arbeitsspeicher, verwenden.

4.3.2.2 Ringbuffer für Sensorwerte

Dieser Ringbuffer, welcher mittels einer Datenbank realisiert wird, wird nun für die einzelnen Sensorgruppen (interne Sensoren, GPS und Sensorboard) eingesetzt. So wird für jede Sensorgruppe eine eigene Datenbank erstellt. In diese drei Datenbanken werden nun jeweils in einen Index eine Zeile der Sensorwerte geschrieben. So werden die Sensordaten der Reihe nach in die entsprechenden Datenbanken abgefüllt.
Es muss nun noch die gebrauchte Grösse der Datenbank ermittelt werden, damit 10 Minuten Sensorwerte in diesem Ringbuffer gehalten werden können.
Da das Programm die internen Variabeln einmal pro Sekunde abfragt, entsteht pro Sekunde eine Zeile und somit in 10 Minuten 600 Zeilen. Somit muss für die Internen Sensoren eine Datenbank mit maximal 600 Einträgen erstellt werden. Analog dazu muss für das GPS eine Datenbank mit maximal 1800 Einträgen erstellt werden(drei Zeilen pro Sekunde) und für die Werte des Sensorboards muss eine Datenbank mit maximal 9000 Einträgen erstellt werden (15 Zeilen pro Sekunde).

Ein Ausschnitt der Datenbank für die Werte des Sensorboardes sieht nun etwa wie in der folgenden Tabelle dargestellt aus:



Index	Inhalt
0	FFA,FDA,FDE,007,841,A4C,A1C,2EA,22A,00D,007,006
1	FFA,FDA,FDE,007,841,A4C,A1C,2EA,22A,00D,007,006
2	FFA,FDF,FDB,006,813,A38,A18,2E5,224,00D,004,007
3	FFA,FE3,FDD,008,7F4,A28,A10,2DF,232,003,007,008
…	…
8997	FFA,FF0,FD8,FFB,7E0,A1B,A0C,2D8,225,00B,001,00D
8998	FFA,FE5,FDD,005,7D4,A1A,A0C,2D4,228,00B,FFB,00B
8999	FFA,FE3,FD4,FFB,7C8,A18,A07,2DD,230,000,004,00C

(Abb. 59: Ringbuffer Datenbank)

Für die Implementierung müssen folgende Schritte beachtet werden. Als erstes muss überprüft werden, ob es die Datenbanken schon gibt, falls ja dann müssen sie geöffnet werden, falls nein müssen sie erst erstellt und dann geöffnet werden.

```
// DB suchen
    gPointerAufDB = DmFindDatabase(0, DBName);

//falls sie schon existiert, DB öffnen
    if (gPointerAufDB >0) {
          gPointerAufGeöffneteDB       =       DmOpenDatabase(0,
DmFindDatabase(0, DBName), dmModeReadWrite);

// Falls sie noch nicht existiert, DB erstellen
    if (!gPointerAufGeöffneteDB) {
          error   =   DmCreateDatabase(0,   DBName,   kCreator,
kDBType, false);
          if (error)
               return error;

// Falls kein Fehler auftrat, DB öffnen
gPointerAufGeöffneteDB  =  DmOpenDatabaseByTypeCreator(kDBType,
kCreator, dmModeReadWrite);
          if (!gPointerAufGeöffneteDB)
               return DmGetLastErr();
```

Um nun in diese Datenbank Daten abfüllen zu können, müssen als erstes die Einträge erstellt werden, dann muss ein Schreibschutz auf den betreffenden Eintrag gelegt werden, damit nicht ein anderes Programm gleichzeitig auf diesen Eintrag zugreifen und ihn verändern kann, da dies zu Fehlern führen würde. Hat man nun den Schreibschutz aktiviert, kann der Eintrag beschrieben werden. Sobald fertig geschrieben ist, muss der Eintrag wieder freigegeben werden, damit er anderweitig wieder verwendet werden kann. Zu beachten gilt auch, dass wenn der Ringbuffer einmal durchgelaufen ist, dass die Einträge nicht mehr neu erstellt werden, sondern nur noch überschrieben werden müssen. Das heisst, dass dann keinen neue Memory-Allokationen durchgeführt werden müssen und somit die Zuweisung schneller läuft.

```
//Falls Eintrag noch nicht existiert

if(numRecs < (Zaehler + 1)){
    //Erstelle einen neuen Eintrag mit dem  Index Zaehler
    recH = DmNewRecord(gDB, &Zaehler, StrLen(record-
>strings)+1);
               if (recH) {
               //blockiere den Eintrag
                     recP = MemHandleLock(recH);
               //Schreibe die Daten in den Eintrag
                     DmWrite(recP, 0, record->strings,
StrLen(record->strings)+1);
               //Eintrag wieder freigeben
                     MemHandleUnlock(recH); }}
```

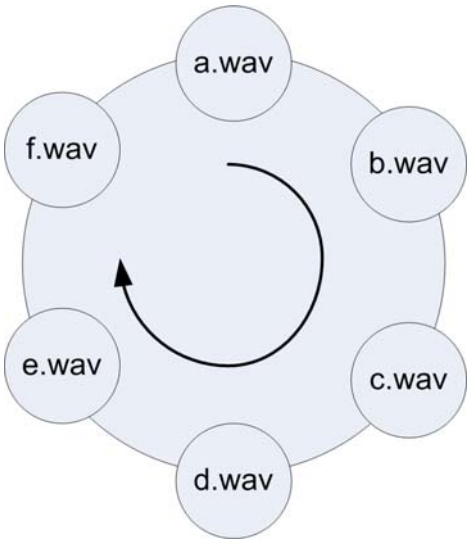
```
//Falls der Eintrag schon existiert / Ringbuffer schon mind.
//Einmal rundherum
if (numRecs >= (Zaehler + 1)){
    //Suche Eintrag
    recH = DmGetRecord(gDB,Zaehler);
               if (recH) {
               //blockiere den Eintrag
                     recP = MemHandleLock(recH);
               //Schreibe die Daten in den Eintrag
                     DmWrite(recP, 0, record->strings,
StrLen(record->strings)+1);
               //Eintrag wieder freigeben
                     MemHandleUnlock(recH);
               }
}
```

Das Auslesen der Einträge funktioniert etwa gleich wie das Schreiben. Als erstes muss der Eintrag gesucht, dann blockiert, gelesen und anschliessend wieder freigegeben werden. Beim Auslesen muss, wie im letzen Kapitel bereits erwähnt, darauf geachtet werden, dass die Reihenfolge, in welcher die Einträge ausgelesen werden, stimmt.
Zur Überprüfung des Ringbuffers wurde, für jede Sensorgruppe einzeln, parallel in den Ringbuffer und in eine Textdatei auf der SD-Karte geschrieben und die Werte anschliessend verglichen. So kann Ausgeschlossen werden, dass beim schreiben und lesen des Ringbuffers ein Fehler passiert.
Zusätzlich muss auch beachtet werden, dass sobald ein Telefonanruf eingegangen und die dazugehörigen Daten auf die SD-Karte gespeichert sind oder wenn das Programm normal beendet wird, die Datenbanken alle komplett gelöscht werden. Würde man sie nicht löschen, wären evtl. noch alte Daten in den Datenbanken gespeichert, was zu fast unauffindbaren Datenfehlern führen würde. Hinzu kommt, dass wenn ein Eintrag neu erstellt werden möchte, welcher aber bereits schon existiert, dies zu einem Absturz des Telefons führen kann. Zusätzlich werden die drei Datenbanken nach jedem Neustart des Treo’s gelöscht und wieder neu erstellt, für den Fall eintreten, dass während des Telefongesprächs der Akku leer ist und das Telefon sich ausschaltet, bevor die Datenbanken geleert werden können.

4.3.2.3 Ringbuffer für Audio

Da die Audioaufnahme nicht in einem konstanten Intervall immer wieder eine Linie von Sensorwerten schickt, sondern ein kontinuierlicher Stream von Bytes ist, ist es nicht möglich, für die Audioaufnahme das Datenbanksystem des Treo's zu verwenden, da die einzelnen Einträge der Datenbank nur eine beschränkte Grösse haben können.
Die Aufnahme würde mit ihren 4,8 MB ziemlich viel Speicher belegen, das ginge zwar noch, aber dadurch, dass für das Öffnen, Locken und wieder Freigeben eines Eintrages geht mindestens soviel Zeit verloren, wie für das Schreiben des Eintrages selbst, ginge bei einem kontinuierlichen Datenstrom ca. 50% verloren.

Als Alternative wird für den Ringbuffer für die Audioaufnahme direkt Dateien auf der SD-Karte verwendet. Auf der SD-Karte werden in einem temporären Ordner sechs Wave Dateien angelegt. (a.wav; b.wav; c.wav; d.wav; e.wav und f.wav) Wenn das Programm gestartet ist, wird in Datei „a.wav" der „Audiostream" gespeichert. Sobald in einem Timer[16] zwei Minuten abgelaufen sind, wird die Aufnahme gestoppt und wieder begonnen, diesmal ist das Ziel für den „Audiostream" aber die Datei „b.wav". So wird alle zwei Minuten die nächste Datei gefüllt. Bis nach 12 Minuten die Datei „f.wav „voll ist, danach wird wieder die Datei „a.wav" überschrieben.



(Abb. 60:Ringbuffer mittels Dateien)

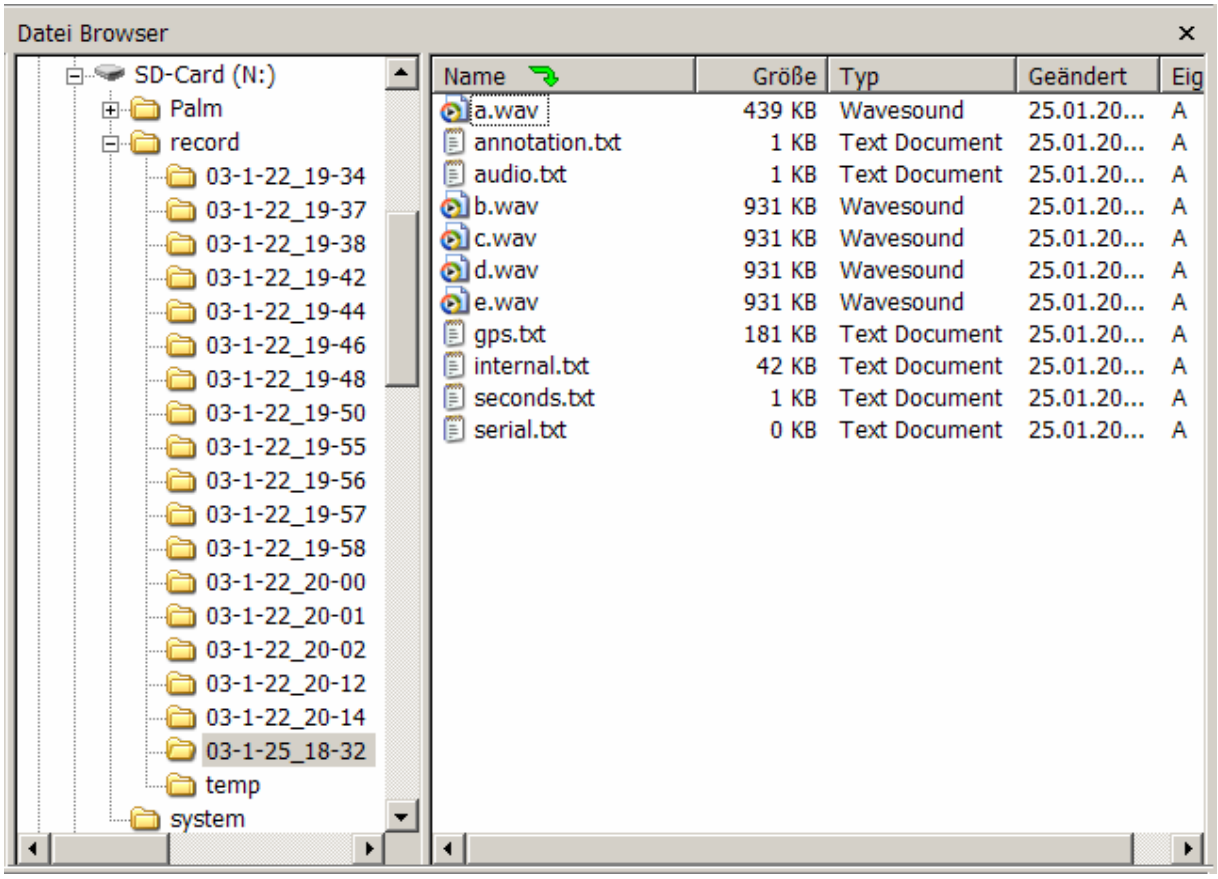
Es werden bewusst sechs Dateien verwendet. Denn falls ein Anruf eingeht, kurz nachdem wieder in eine neue Datei gespeichert wird, so ist in dieser Datei evtl. nur wenige Sekunden abgespeichert, aber die restlichen fünf Audiodateien zusammengesetzt ergeben wieder die benötigten 10 Minuten.
Ein kleiner Nachteil ist, dass durch das Stoppen und wieder Anfangen der Audioaufnahme bis zu 2 Sekunden zwischen den einzelnen Teilen verloren gehen. Diese Ungenauigkeit liegt aber im Vergleich zur der gesamten Aufnahmezeit noch in der Toleranzgrenze.

[16] Als Timer wird hier der schon bestehende Timer verwendet, welcher für die Neukalibrierung der Magnetfeldsensoren alle 2 Minuten gebraucht wird.

4.3.3 Ordnerstruktur

Um die Sensordaten, welche in dem Ringbuffer gehalten werden, geordnet abspeichern zu können, ist es nötig eine übersichtliche Ordnerstruktur anzulegen. Für diesen Zweck ist auf der SD-Karte ein Ordner mit dem Name „Record“. In diesem Ordner werden wiederum einzelne Ordner, die mit Datum und Zeit des Anrufes versehen sind, erstellt, welche alle Sensordaten zu dem zugehörigen Anruf enthalten.
Sobald das Programm zur Sensoraufnahme gestartet wird, wird ein temporärer Ordner mit dem Name „temp“ erstellt. In diesen Ordner werden die Dateien erstellt, in welche der "Audiostream“ geleitet wird. So befinden sich nach 10 Minuten in diesem Ordner die fünf Audiodateien „a.wav,b.wav,c.wav,d.wav und e.wav“. Die restlichen Sensordaten befinden sich in der als Ringbuffer verwendeten Datenbanken.
Sobald nun ein Anruf eingeht, wird die Zeit und das Datum zwischengespeichert. Ist nun das Gespräch beendet und der Fragebogen nach dem Anruf ausgefüllt, wird der Ordner „temp“ zu einem Ordner mit dem Name des zwischengespeicherten Timestamps umbenannt. Dieser hat die Form JJ-MM-DD_HH-MM (Jahr-Monat-Tag_Stunden-Minuten).
Zusätzlich zu in diesem Ordner sich bereits befindenden Audiodateien wird eine Textdatei mit dem Name „audio.txt“ geschrieben. In dieser Datei wird notiert, welche der Audiodateien, die erste ist, damit sie später in der richtigen Reihenfolge wieder zusammengesetzt werden können. In der Datei „seconds.txt“ steht als Zusatzinformation die gesamte Dauer der Aufnahme. Wenn die Dauer der Sensoraufnahmen von dieser Zahl abweicht, lässt sich daraus schliessen, dass irgendein Aufnahmeausfall sich ereignete. In die Datei „annotation.txt“ werden die Ergebnisse der Befragung durch das Menu gespeichert.
Sind diese Textdateien alle gespeichert, werden nun die Datenbanken geöffnet, in der richtigen Reihenfolge ausgelesen und anschliessend wieder geschlossen. Die Inhalte der Datenbanken werden ebenfalls in Textdateien gespeichert. Die Werte des Sensorboards finden sich in der Datei „serial.txt“, die Werte des GPS-Empfängers sind in der Datei „gps.txt“ und die Werte der internen Variabeln des Telefons werden in der Datei „internal.txt“ gespeichert.

Die Ordnerstruktur und der Inhalt eines Ordners zu einem Telefonevent sind auf der folgenden Graphik ersichtlich.



(Abb. 61: Ordnerstruktur zur Speicherung)

Wenn das Programm normal beendet wird, sprich, wenn der Benutzer ein anderes Programm startet, wird der Ordner „temp“ und sein ganzer Inhalt gelöscht, damit bei einer neu gestarteten Aufnahme die alten und neuen Daten vermischt werden.

Auf den genaueren zeitlichen Ablauf des Speicherns und dessen Integration in das Programm wird im Kapitel 4.4.4 „Ablauf der Speicherung“ noch genauer eingegangen.

4.4 Ablauf des Programms

Wie bereits erwähnt, ist das Betriebssystem PalmOS 5.2 Garnet nur beschränkt multithreading fähig. Diese Tatsache und die erwünschte Funktionalität des Programms erfordern, dass die Applikation seriell abgearbeitet wird.
Der gewünschte serielle und zeitlicher Ablauf des Programms lässt sich mit Hilfe der Launchcodes (Kapitel 3.2.4.1) und der Notificationen (Kapitel 3.2.4.2) realisieren, da diese dem Programm die Möglichkeit geben, auf einen eintretenden Event entsprechend reagieren zu können.

Dieser Aufbau des Programms erwies sich als sehr komplex, da jeder mögliche Ablauf berücksichtigt und abgehandelt werden musste, und als sehr kompliziert, da es nicht für jede Situation eine entsprechende Notification gibt.

Um den gesamten seriellen Ablaufes des Programms so einfach wie möglich darstellen zu können, setzte ich diesen Ablauf nun Schritt für Schritt zusammen. Die folgenden Kapitel 4.4.1 und 4.4.2 sind sehr technisch geschrieben und mit viel Programmcode versehen. Dies wurde bewusst so gewählt, da diese Kapitel eine sehr zentrale Bedeutung für die Implementierung des gesamten Programms haben.

4.4.1 Laufen des Programms sicherstellen

Da das Programm für ein Langzeitexperiment ausgelegt ist, ist es wichtig, dass der Benutzer das Telefon möglichst uneingeschränkt benutzen kann und wenig von dem Aufnehmen der Daten mitbekommt. Am einfachsten wäre dies realisierbar, wenn man einen eigenen Thread im Hintergrund zur Aufnahme der Sensordaten starten könnte. Dies ist aber durch das Singelthread Betriebssystem nicht möglich. Bedingt durch diese Einschränkung bleiben nur noch zwei mögliche Ansätze. Diese sind:

- Man lässt das Programm immer laufen, somit kann das Telefon für nichts anderes mehr verwendet werden.
- Man lässt zu, dass das Telefon auch anderweitig benutzt werden kann und nimmt dafür in Kauf, dass die Aufnahmen von Zeit zu Zeit unterbrochen werden.

Die erste Variante wird umgesetzt, dadurch, dass man das Programm zur Sensoraufnahme immer laufen lässt, indem man den Versuch, das Programm zu beenden, abfängt und das Sensoraufnahmeprogramm wieder startet. Dies ist möglich, indem man die „Notifiaction“ „`sysNotifyAppQuittingEvent`“ abfängt und mit `SysAppLaunch` das Programm wieder neu startet. Alternativ wäre es auch möglich, die Hardwarebuttons zu deaktivieren. So würde der gleiche Effekt erzeugt und das Sensoraufnahmeprogramm kann nicht beendet werden. Dies wird mit dem Befehl `HsKeyEnableKey (Tastenname, false);` erreicht. Die Namen aller Tasten lassen sich in der „Headerdatei“ `HSKeyCodes.h`, welche in dem SDK enthalten ist, nachschlagen.

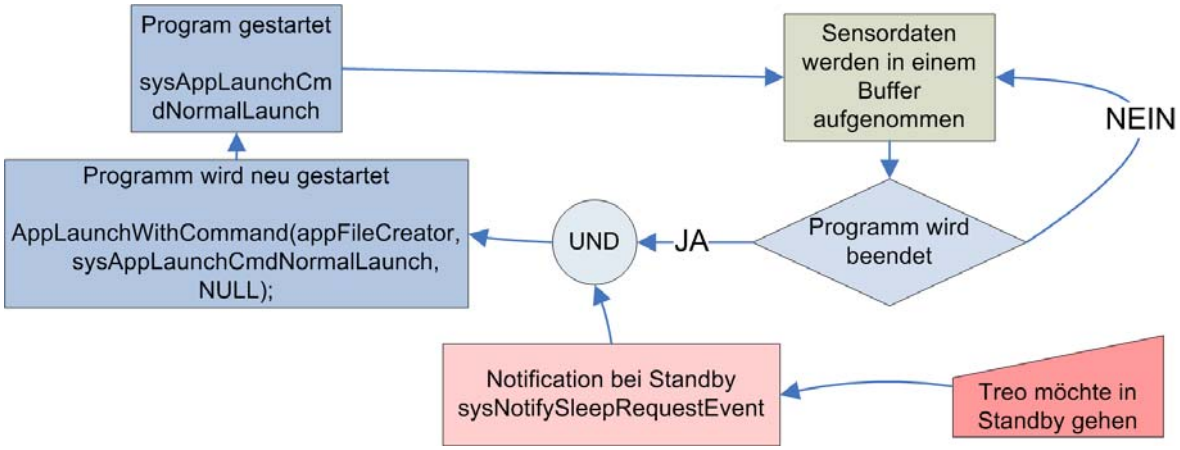
Dieser Ansatz stellt zwar sicher, dass immer die Sensordaten ununterbrochen aufgenommen werden, schränken den Benutzer des Telefons aber extrem ein. Fast alle Funktionen des Telefons wären so nicht mehr zugänglich und der Benutzer könnte nicht einmal mehr selbst jemanden anrufen, da er die Telefonapplikation nicht mehr starten kann. Das Telefon könnte so lediglich Sensorwerte aufnehmen und Angerufen werden. Dies wären, aufgrund der lückenlosen Aufnahmen, für das geplante Experiment zwar sehr gute Voraussetzungen, doch würde das Experiment sehr wahrscheinlich an der Benutzerakzeptanz scheitern. Somit kommt diese Variante nicht in Frage.

Die zweite Variante ist es, in Kauf zu nehmen, dass nicht immer die Sensorwerte aufgenommen werden und so dem Benutzer die Möglichkeit gelassen wird, das Sensoraufnahmeprogramm zu beenden, so dass er nun die ganze Funktionalität des Telefons benutzen kann.

Um dies zu realisieren, muss das Programm erst einmal gestartet werden. Als geeigneter Einstiegspunkt bietet sich der Event an, in welchem ein Reset oder eine Synchronisation des Telefons gestartet wird. Diese beiden Ereignisse können mit den „Notificationen“ „`sysAppLaunchCmdSystemReset`“ „`sysAppLaunchCmdSyncNotify`„ abgefangen werden. An diesem Punkt ist es möglich unser Programm für die nötigen „Notificationen“ zu registrieren um den weiteren Ablauf des Programms mittels den entsprechenden Notificationen steuern zu können.

Ist der Reset oder die Synchronisation abgeschlossen, kann dies mit den „Notificationen“ „`sysNotifyResetFinishedEvent`“ und „`sysNotifySyncFinishEvent`“ abgefangen werden. Bevor auf diesen Event folgend unser Programm gestartet wird, muss sichergestellt werden, dass die Telefonfunktion des Treo’s aktiviert ist, damit eingehende Anrufe auch ankommen. Die Telefonapplikation wird mit dem API-Aufruf `HsTurnRadioOn ();` gestartet. Der Benutzer kommt nun nach dem Start des Telefons direkt zur Eingabe seines Pincodes. Ist dieser eingegeben, wird das Programm zur Sensoraufnahme gestartet.

Läuft das Programm einmal, gestaltet sich der Ablauf des Programms wie im folgenden Diagramm dargestellt ist:



(Abb. 62: Neustart des Programms)

Das Programm befindet sich in einer Endlosschleife, in welcher es die Sensordaten in einen Buffer aufnimmt. Wird nun ein anderes Programm wie der Kalender, Adressbuch oder auch die Telefonapplikation gestartet, wird das Aufnahmeprogramm beendet und der Benutzer kann das neu gestartete Programm verwenden. Sobald der Benutzer 30 sek. lang nichts mit dem Telefon macht, geht das Telefon automatisch in den „Standby-Modus“. Dieser Event kann nun ausgenützt werden um unser Programm wieder zu starten. Der Event, dass das Telefon in den „Standby-Modus“ wechselt, lässt sich abfangen mit der „Notification“ „`sysNotifySleepNotifyEvent` „. Dies ist aber zu spät, denn wenn nach dieser „Notification“ das Programm gestartet wird, startet das Programm erst nachdem das Telefon wieder manuell aus dem „Standby-Modus“ herausgeholt wird. Deshalb muss der Event, bevor das Telefon in den „Standby-Modus“ geht, abgefangen werden. Das ist mit der „Notification“ „`sysNotifySleepRequestEvent` „ möglich.

Diese wird gesendet, sobald das Telefon plant in den „Standby-Modus“ zu wechseln. Mit Hilfe dieser „Notification“ lässt sich nun das Programm wieder starten und dadurch, dass der Vorgang, dass das Telefon in den „Standby-Modus“ gehen möchte, unterbrochen wurde, bleibt dem Programm genug Zeit (30 Sek.) um sauber zu starten und mit den Aufnahmen zu beginnen, bevor das Telefon wieder in den „Standby-Modus“ wechselt und die Sensordaten aufnimmt, bis entweder das Programm wieder durch den Benutzer unterbrochen wird oder ein Telefonanruf eingeht.

Der Programmcode um das Programm wieder zu starten, sobald die Notification `sysNotifySleepRequestEvent` eingeht und das Programm noch nicht läuft sieht folgendermassen aus:

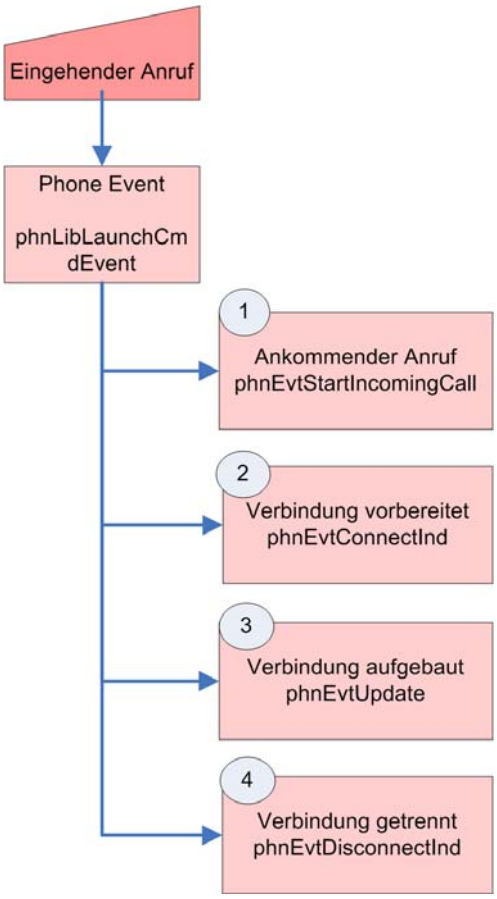
```
//NotificationLaunchcode
if (cmd == sysAppLaunchCmdNotify) {
SysNotifyParamType* paramP = (SysNotifyParamType*) cmdPBP;

//Notification sysNotifySleepRequestEvent abfangen
if(paramP->notifyType == sysNotifySleepRequestEvent){

      //Falls das Programm nicht läuft
      if (!(launchFlags & sysAppLaunchFlagSubCall)){

      //Programm Starten
    AppLaunchWithCommand(appFileCreator,
sysAppLaunchCmdNormalLaunch,NULL);
          }
      }
}
```

4.4.2 Anrufabarbeitung, bei aktiver Applikation

Bevor ich den genauen Ablauf des Abfangens und Handelns auf einen eingehenden Anruf erläutere, möchte ich die involvierten „Launchcodes“ aufzeigen. Es gibt zwar eine „Notification“ „`kTelTelephonyNotification`“, welche bei Telefonevents verschickt werden sollten, doch ist diese nur von PalmSource vorgesehen und wurde von PalmOne, den Entwicklern des Treo’s, nicht umgesetzt und steht uns somit nicht zur Verfügung. PalmOne bietet aber als Ersatz einen Launchcode, welcher diese Funktionalität übernimmt. Dieser Launchcode heisst „`phnLibLaunchCmdEvent`“ und wird aufgerufen, sobald irgendein Event von der Phone Library registriert wird. Dies setzt voraus, dass die Phone Bibliothek zuvor geöffnet und das Programm für Phone Events registriert wurde. In unserem Programm wird dies in der Startfunktion, welche bei Beginn des Programms aufgerufen wird, abgehandelt.

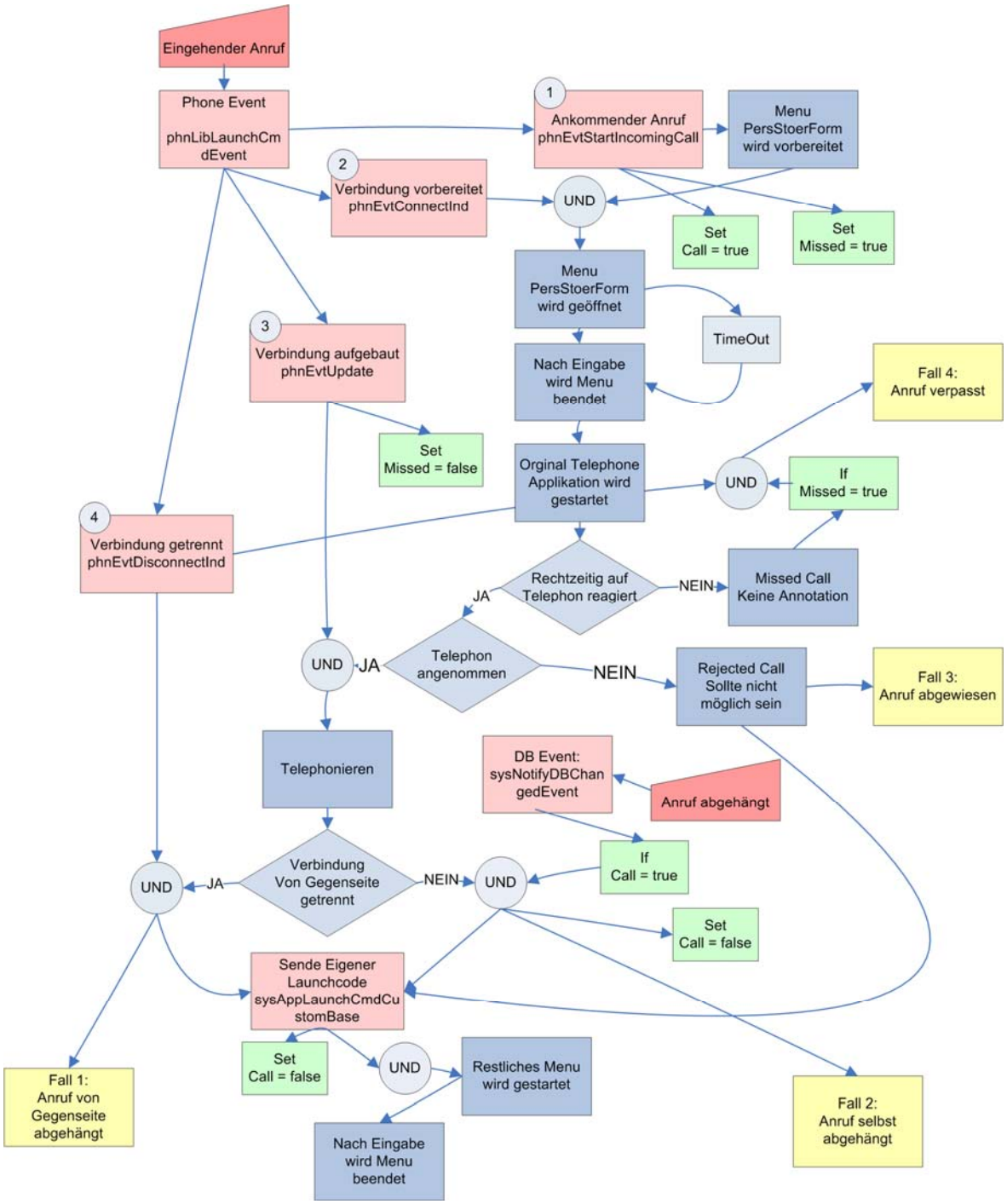
```
// System Bibliothek öffnen

err = HsGetPhoneLibrary (&PhoneLibRefNum);
    if (err){
         FrmAlert (resAlertPhoneLibrary);
    }
    else{
    // Unser Programm für Phone Events registrieren

      err = PhnLibRegister(PhoneLibRefNum, 'TRER',
phnServiceVoice);
         if (err)  FrmAlert (resAlertPhoneLibrary);
    }
```

Ist in unserem Programm nun die Bibliothek geöffnet und für die Events registriert, wird jedes Mal, wenn ein mit der PhoneLibary zusammenhängender Event eintrifft, dem Programm der Launchcode „`phnLibLaunchCmdEvent`“ zugeschickt. Diesem Launchcode wird zusätzlich ein Parameter mitgegeben, welcher es möglich macht, die einzelnen Events zu identifizieren und zu unterscheiden. Eine komplette Liste dieser Parameter findet sich im Anhang und genauere Erläuterungen zu diesen finden sich in der „Headerdatei“ „`HsPhoneEvent.h`“, welche dem PalmOneSDK beiliegt. An dieser Stelle möchte ich nur die vier Parameter erwähnen, welche für unser Programm von Bedeutung sind. Die Reihenfolge, in welcher diese vier Events auftreten, ist in folgendem Diagramm dargestellt.



(Abb. 63: Launchcodes bei eingehendem Anruf)

Sobald in der „PhoneLibary“ registriert wird, dass ein eingehender Anruf ansteht, wird der „Launchcode“ „`phnLibLaunchCmdEvent`“ mit dem Parameter „`phnEvtStartIncomingCall` “[1] an unser Programm gesendet. Sobald das Telefon bereit ist eine Verbindung aufzubauen wird wieder derselbe „Launchcode“, nun aber mit dem Parameter „`phnEvtConnectInd`“[3], an das Programm geschickt. Wird nun das Gespräch angenommen und die Verbindung ist aufgebaut wird der Parameter „`phnEvtUpdate`“ [3] dem „Launchcode“ mitgeliefert. Beendet der Anrufer die Verbindung, wird der Parameter „`phnEvtDisconnectInd`“ [4] übermittelt.
Die Schwierigkeit liegt nun darin, jeden möglichen Fall korrekt abzuhandeln, da z.B. wenn ein Anruf verpasst wird, wohl auch Schritt [1] und [2] durchgegangen werden. Da aber nie eine Verbindung aufgebaut wird, springt er, in diesem Fall, dadurch direkt zu Schritt[4] und überspringt Schritt[3]. Für den Fall, dass der Angerufene selbst die Verbindung unterbricht, existiert kein entsprechender Parameter.
Da der aus diesen Bedingungen sich ergebende Ablauf etwas komplex ist, habe ich den Ablauf im folgenden Diagramm dargestellt.



(Abb. 64: Verlauf der 4 Fälle)

Der Ablauf lässt sich in vier Fälle einteilen, welche in dem Diagramm als gelbe Rechtecke dargestellt sind. Diese ergeben sich aus den Situationen, wenn der Anruf selbst beendet, wenn er von der Gegenseite beendet, wenn er abgewiesen oder wenn er verpasst wird. Bevor ich auf diese vier Fälle einzeln eingehe, möchte ich noch ein grundlegendes Problem und dessen Lösung erwähnen.
Da durch die „Launchcodes“ immer ein von dem restlichen Programm unabhängiger Programmteil gestartet wird, können zwischen diesen Programmteilen keine globalen Variabeln verwendet oder übergeben werden. (Kapitel 3.2.4.1). Für den Ablauf unserer Applikation ist es aber notwendig, um die verschiedenen Fälle unterscheiden zu können, dass wir die Möglichkeit haben gewisse Zustände zwischen zuspeichern. Um dieses Problem zu umgehen, gibt es zwei Möglichkeiten. Die erste wäre eine eigene Datenbank

zu erstellen und jeder benötigten Variabeln einen eindeutigen Index zuzuordnen. So ist es möglich, den Wert der Variable in die Datenbank zu schreiben und in einem anderen Programmteil wieder diese Datenbank zu öffnen und den Wert der Variablen wieder auszulesen. Diese Variante wäre zwar effizient, scheitert aber daran, dass die Datenbank immer wieder geöffnet und geschlossen werden muss, damit sie im anderen Programmteil nicht blockiert ist. Da die Datenbank aber relativ lange zum Schliessen braucht, ist es möglich, dass ein Programmteil sie versucht zu öffnen während sie noch nicht richtig beendet ist. Dies kann unter Umständen einen Absturz verursachen. Aus diesen Stabilitätsgründen kommt diese Variante nicht in Frage.
Die zweite Möglichkeit ist es, für jede Variable auf der SD-Karte eine Datei zu erstellen und den Wert der Variabel in diese Datei zu schreiben, so kann an jeder beliebigen Stelle des Programms der Wert der Variabeln aus der Datei gelesen werden. Dies ist im Vergleich zur ersten Variante zwar etwas langsamer, da immer wieder auf die SD-Karte zugegriffen werden muss, aber sie läuft dafür absturzsicher.
Um diese Dateien sauber aufgeräumt zu haben, wird vom Programm ein Ordner mit dem Namen „System“ erstellt. In diesen Ordner werden alle Dateien abgelegt, welche als Zwischenspeicher für globale Variabeln dienen.
Um den Ablauf eines eingehenden Anrufes in die oben genannten vier Fälle einteilen zu können, genügt es, wenn wir zwei Variabeln in je einer Datei zwischen speichern. Beide Variabeln sind „Booleans“ und können nur „true“ oder „false“ ein. Die Eine wird in die Datei „Call“ geschrieben. Steht sie auf „true“ bedeutet dies, dass das Ende eines eingehenden Anrufes noch nicht abgefangen worden ist. Die zweite wird in die Datei „Missed“ gespeichert und steht auf „true“, solange der Anruf nicht angenommen wurde, d.h. dass der Anruf verpasst worden ist. Diese Werte der Variabeln sind in der obigen Abbildung in grünen Rechtecken dargestellt.
Mit Hilfe dieser beiden Variabeln gelingt es die genannten vier Fälle zu unterscheiden. Ich möchten nun auf diese Fälle einzeln und der Reihe nach eingehen.

4.4.2.1 Fall 1: Anruf angenommen und von Gegenseite beendet

Dies ist der einfachste Fall und zudem ist dieser Fall auch der einzige, welcher problemlos umsetzbar war. Der Ablauf sieht nun folgendermassen aus: Sobald ein eingehender Anruf registriert wird, setzt das Programm die beiden Variabeln „Call“ und „Missed“ auf „true“ und bereitet das erste Menu vor. An diesem Punkt darf das Menu noch nicht geöffnet werden, da es sonst von dem nächsten Launchcode wieder unterbrochen wird und das Telefon abstürzt. Wird nun der nächste Launchcode gesendet und das Telefon wäre bereit eine Verbindung aufzubauen, wird der Bildschirm des Treo’s aktiviert, die Tastensperre aufgehoben, die Tastaturbeleuchtung eingeschaltet und das erste Menu geöffnet.

```
//Bildschirm aktivieren
EvtResetAutoOffTimer ();

//Tastaturbeleuchtung einschalten
keyLight = 0;
HsLightMode (true, &keyLight);

// Tastensperre deaktivieren
locked = 0;
HsAttrSet(hsAttrKeyboardLocked ,0,&locked);
```

Währendem das Menu auf dem Bildschirm ist, macht das Telefon zusätzlich durch einen, sich immer wieder wiederholenden, „Systembeeb“ und den Vibrationsalarm auf sich aufmerksam.

```
//Vibration
HsIndicatorState
(vibrateCount,kIndicatorTypeVibrator,&vibrate);
```

Damit dieser improvisierte Klingelton nicht zu leise ist, wird kurz vor dem eingehenden Anruf die Lautstärke der Systemtöne auf die maximale Lautstärke gestellt.

```
//systemsound Lautstärke einstellen
    SndSetDefaultVolume (NULL,SystemSoundP, NULL);
```

Als Alternative gibt es einen API-Aufruf, mit welchem man eine beliebige MP3 Datei im Hintergrund abspielen kann[17]. Dies wurde aber nicht weiter verfolgt, da die Priorität auf andere Aspekte gesetzt wurde.

Nachdem das erste Menu ausgefüllt ist, wird dieses beendet und die eigene Telefonapplikation des Treo’s gestartet. Mit dieser kann der Benutzer nun das Telefonat entgegennehmen. Sobald das Gespräch beendet ist und der Anrufer das Telefon aufhängt und somit die Verbindung beendet, wird unserem Programm der Launchcode, dass die Verbindung getrennt wurde, zugeschickt. An dieser Stelle schickt sich das Programm einen selbst definierten Launchcode, mit welchem nun das zweite Menu gestartet wird. Es wäre möglich, dies auch ohne einen selbst definierten Launchcode zu machen, doch vereinfacht dies den Ablauf des nächsten Falles. Ist das Menu fertig ausgefüllt, wird wieder der Programmteil, welcher die Sensorwerte aufnimmt, gestartet.

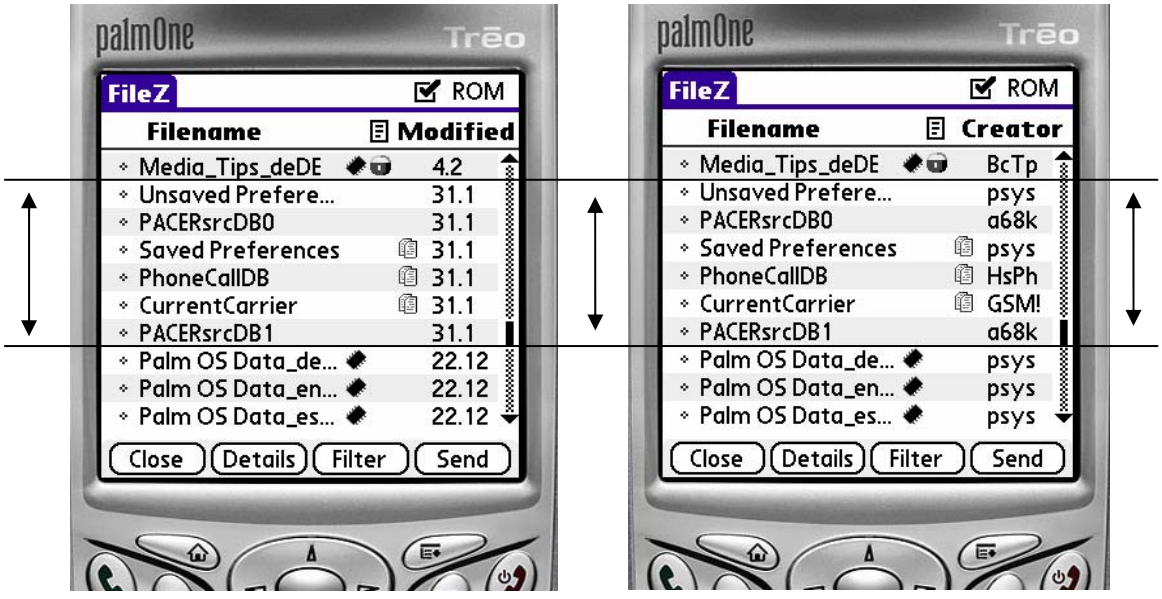
[17] Die Audiofunktionen sind die einzigen Feature, welche im Hintergrund, parallel zu anderen Aufgaben ausgeführt werden können

4.4.2.2 Fall 2: Anruf angenommen und selbst beendet

In diesem Fall ist der Ablauf genau gleich wie im ersten Fall, bis zu dem Punkt, an welchem die Verbindung beendet wird. Hängt nun der Benutzter selbst auf, wird kein Launchcode geschickt. Dies bedeutet, dass dieser Fall nicht mit den von den Entwicklern vorgesehenen Mitteln abgefangen werden kann. Um für diesen Fall trotzdem eine Möglichkeit zu finden, ihn abhandeln zu können sind folgende Überlegungen getroffen worden:
Wenn ein Telefonat beendet ist, wird dies in der Telefondatenbank unter den letzten Anrufen gespeichert. Diese Tatsache habe ich mir zu nutzen gemacht und den „Event“, dass etwas in die entsprechende Datenbank gespeichert wird, abgefangen. Um die Datenbank zu finden, habe ich das Freewaretool FileZ [nosleep, 2006] herunter geladen und installiert. Dieses Programm ist ein Dateimanager. Mit Hilfe diese Tools habe ich als erstes alle Datenbanken anzeigen lassen.
Alle Datenbanken, die auf dem Treo verfügbar sind, werden hier grün dargestellt.

(Abb. 65: Datenbanken auf Treo)

Anschliessend habe ich das Datum des Treo’s auf den 31.1 gesetzt und mich selbst angerufen um anhand des Timestamps beobachten zu können, welche Dateien durch den Anruf verändert worden sind.



(Abb. 66: Veränderte Dateien nach Anruf)

Für unseren Zweck spring die Datei „PhoneCallDB“ sofort ins Auge, doch diese Datei ist keine normale Datenbank (Sie ist auch in den oberen Graphiken nicht aufgetaucht) und daher können wir auch keine Veränderung dieser Datenbank abfangen. Durch Probieren hat sich aber herausgestellt, das sich die Veränderung der Datei „Unsaved Perferces“ mittels der Notification „`sysNotifyDBChangedEvent`“ abfangen lässt. Um nicht noch Veränderungen anderer Dateien abzufangen, werden explizit nur Veränderungen von Dateien, welche von dem Creator „psys“ erstellt wurden, überprüft.

```
if(paramP->notifyType == sysNotifyDBChangedEvent){
      if (paramP->broadcaster == 'psys'){
      }
}
```

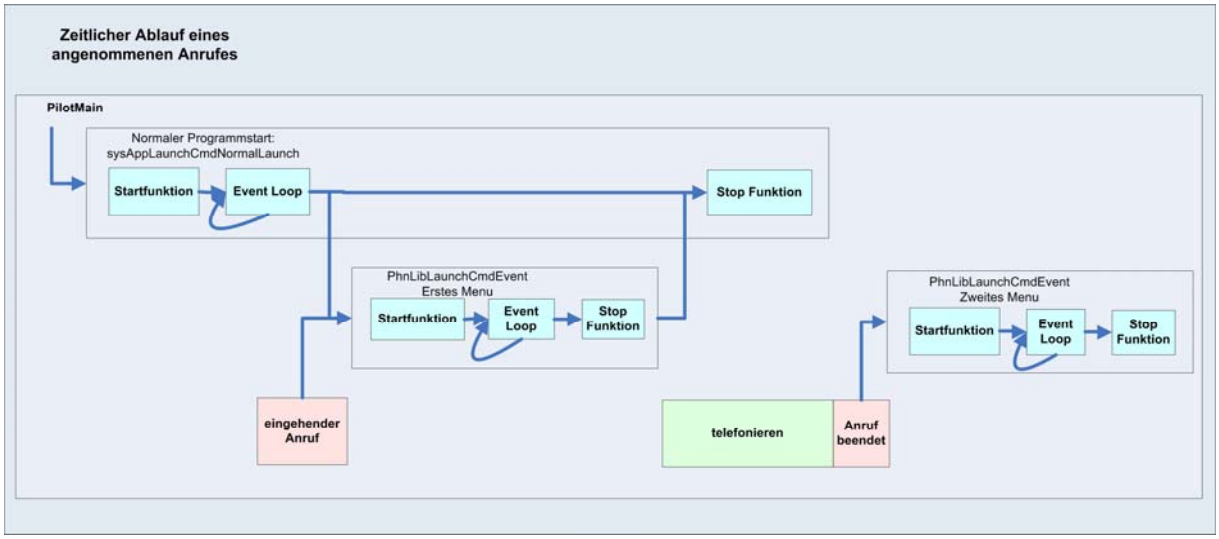
Da diese Datei verwendet wird um noch nicht abgehandelte Einträge zwischen zu speichern, kann der Event, dass diese Datei verändert wird, auch sonst irgendwann auftreten, z.B. wenn ein neuer Kontakt hinzugefügt wird.
Um die Chance eines unerwünschten Events einzuschränken, wird unser Programm für diese „Notification“ erst bei einem eingehenden Anruf registriert und sobald der Anruf abgehandelt ist, sofort wieder unregistriert.
Tritt nun der Event ein, dass diese Datei verändert wurde und die Variable „Call“ steht auf „true“ (d.h. der Anruf wurde noch nicht fertig behandelt, wäre sie auf „false“ wäre der Anruf schon von der Gegenseite beendet worden.), wird wieder der selbst definierte Launchcode dem Programm geschickt und das zweite Menu gestartet.

4.4.2.3 Fall 3: Anruf abgewiesen

Dieser Fall ist eher eine Designfrage. Zu entscheiden galt es, ob der Benutzer überhaupt einen eingehenden Anruf abweisen dürfen soll, oder ob er jeden Anruf entgegen nehmen muss. Einerseits ist es mühsam für den Benutzer, wenn er einen Anruf nicht abweisen kann und es leidet die Benutzerfreundlichkeit des Telefons, aber anderseits kann der Benutzter, bei einem abgewiesenen Anruf keine Aussage machen über den Inhalt und somit auch nicht über die effektive Wichtigkeit des Anrufes. So würden sehr wichtige Informationen verloren gehen. Hinzukommt, dass gerade die Anrufe, welche wirklich stören, die interessantesten für die Auswertung der Unterbrechbarkeit sind, da wir ja das Ablehnen von ungewollten oder störenden Anrufen automatisieren wollen und wir deswegen gerade diese Exempels brauchen um eine Klassifikation machen zu können. Würde man dem Benutzer die Möglichkeit lassen einen Anruf abzuweisen, würde das somit die Resultate evtl. verfälschen. Da die Qualität der Messresultate über die Benutzerfreundlichkeit des Telefons gestellt wurde, fiel die Entscheidung so aus, dass keine Anrufe abgewiesen werden dürfen. Dies wurde umgesetzt indem, sobald die Telefonapplikation des Treo's gestartet wird, der Touchscreen und die Hardwaretaste, um einen Anruf abzuweisen, deaktiviert wurden. So bleibt dem Benutzer keine andere Wahl als mit der Hardwaretaste den Anruf entgegenzunehmen. Somit geht dieser Fall in den ersten oder zweiten bereits erwähnten Fall über.

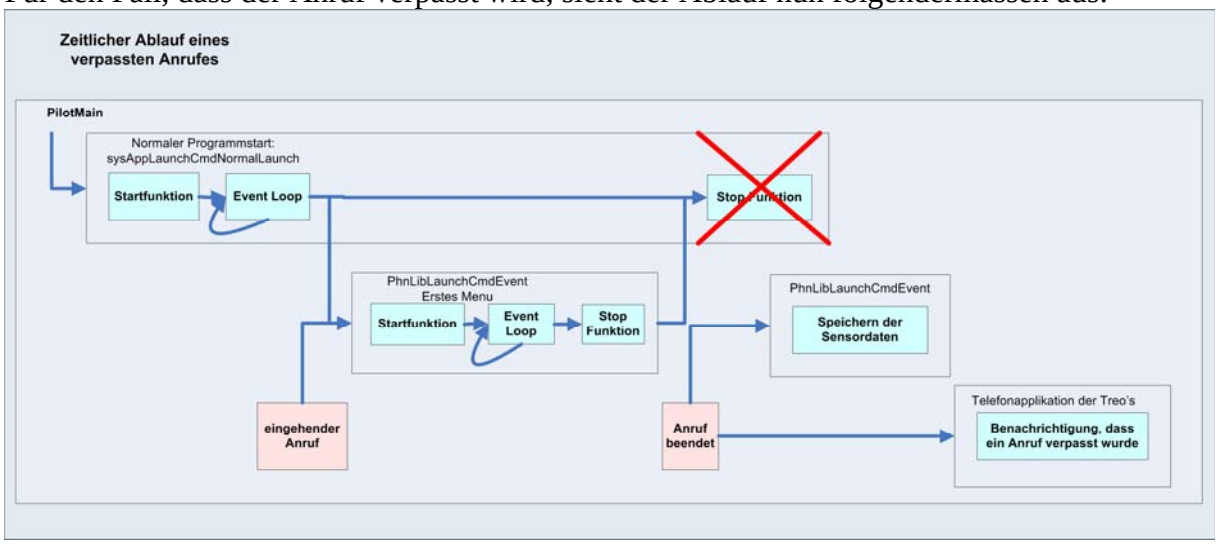
4.4.2.4 Fall 4: Anruf verpasst

Wenn ein Anruf verpasst wurde, ist es meistens für den Benutzter im Nachhinein nicht mehr möglich genau zu beurteilen, ob der Anruf gestört hätte oder nicht. Deshalb haben wir uns dafür entschieden, bei einem verpassten Anruf, zusätzlich zu den gespeicherten Sensordaten, einen Vermerk, dass der Anruf verpasst wurde, hinzuzufügen und gänzlich auf die Annotation der Menus zu verzichten.
Theoretisch ist dieser Fall sehr einfach abzuhandeln doch in der Umsetzung erwies sich dieser Fall als der schwierigste. Bevor ich auf die Problematik dieses Falles eingehe, muss ich zuerst den ersten Fall, dass ein Telefonat angenommen und von der Gegenseite beendet wird, noch einmal aufgreifen.
Der Ablauf der einzelnen Funktionen des gesamten Programms ist in folgendem Diagramm schematisch dargestellt.



(Abb. 67: Ablauf eines angenommenen Anrufes)

Wird unser Programm normal gestartet, durchlauft es erst die „Startfunktion“ und bleibt anschliessend in der Endlosschleife „EventLoop“, in welcher die Sensorwerte aufgenommen werden. Sobald ein Anruf eingeht, hat die Abhandlung des Anrufes höhere Priorität als die „Stop Funktion“, welche zum Beenden des Hauptprogramms benötigt wird. Sobald das erste Menu abgehandelt und das Gespräch angenommen ist, wird diese „Stop Funktion“ noch während dem Telefongespräch nachgeholt und abgearbeitet. Sobald das Gespräch beendet ist, wird wieder das zweite Menu gestartet.
Für den Fall, dass der Anruf verpasst wird, sieht der Ablauf nun folgendermassen aus:



(Abb. 68: Ablauf eines verpassten Anrufes)

Der Anfang verläuft genau gleich. Der Sensoraufnahmeprogrammteil wird beendet und die Telefonapplikation hat wieder die höhere Priorität als die „Stop Funktion“. Während dem ersten Menu ist ein Timer eingebaut, welcher das Menu nach einer gewissen Zeit von selbst schliesst, so kann auch in diesem Fall wieder die Telefonapplikation des Treo’s übernehmen. Wird der Anruf nun aber verpasst, wird der gleiche Event gesendet, wie wenn der Anrufer das Gespräch beendet hätte. Mit Hilfe der Variabeln „missed“ lässt sich der Fall, dass der Anruf von der Gegenseite terminiert wurde und der Fall, dass der Anruf verpasst wurde, unterscheiden. Das Problem liegt aber darin, dass der Treo nun die „Stop Funktion“ abhandeln sollte, die Sensorwerte noch speichern muss (auf den genauen Ablauf der Speicherung wird im nächsten Abschnitt eingegangen) und gleichzeitig versucht einen Benachrichtigungsbildschirm zu öffnen, um anzuzeigen, dass ein Anruf verpasst wurde. Aber bekanntlich ist der Treo ja nicht Multitasking fähig. Die „Stop Funktion“ wird vom Treo einfach verworfen und nie mehr abgehandelt. Die meisten Befehle dieser Funktion liessen sich in das Ende des „Event Loops“ emigrieren, doch bleibt der Befehl, welcher alle Bildschirme des Hauptprogramms schliesst und sie aus dem Arbeitsspeicher entfernt, unabgehandelt. Der Benachrichtigungsbildschirm wartet zwar schön bis die die Speicherung erledigt ist, doch stürzt das Telefon ohne spezielle Behandlung an dieser Stelle unweigerlich ab, da es zu einem Speicherkonflikt mit den sich noch im Speicher befindenden Bildschirminhalten des Sensoraufnahme -Programmteils kommt.
Durch langes Ausprobieren und mit etwas Glück habe ich entdeckt, dass durch Überschreibung des Speichers durch eine eigens definierte Fehlermeldung und durch das anschliessend manuelle Starten der Telefonapplikation mit der „verpassten Anrufe Ansicht“ dieses Problem umgehen und sich so der Absturz vermeiden lässt.

Der exakte Programmcode um das Problem zu umgehen sieht folgendermassen aus:

```
//Telefondatenbank öffnen
DmGetNextDatabaseByTypeCreator(true, &searchState,
sysFileTApplication, hsFileCPhone, true, &cardNo, &dbID);

//Manuelles Starten der Telefon Applikation in Verpasster
//Anrufe ansicht
err = SysUIAppSwitch(
cardNo,dbID,phoneAppLaunchCmdViewMissedCalls ,NULL);

//Eigene Fehlermessage
    FrmCustomAlert(MissedCall,"","","");

//Nochmals manuelles Starten der Telefon Applikation in
//Verpasster Anrufe Ansicht

err = SysUIAppSwitch(
cardNo,dbID,phoneAppLaunchCmdViewMissedCalls ,NULL);
```

4.4.3 Anrufabarbeitung, bei inaktiver Applikation

Die Ausführungen des Kapitels 4.4.2 beschreiben den Ablauf des Programms, währenddem es aktiv ist. Beendet nun aber der Benutzer das Programm um z.B. einen Termin in den Kalender einzutragen, wird die Sensoraufnahme gestoppt. Geht nun ein Anruf ein, währenddem das Programm zur Aufnahme nicht läuft, macht es keinen Sinn diesen Event zu speichern, da zu ihm keine Sensorwerte vorhanden sind.
Um dem Benutzer aber möglichst immer die gleiche Ansicht zu bieten, wird das erste Menu vor dem Abnehmen des Anrufes eingeblendet. So muss er diese Frage beantworten, egal ob die Applikation zur Sensoraufnahme gelaufen ist oder nicht. Seine Antwort wird aber in diesem Fall nicht gespeichert. Auf das zweite Menu nach dem Anruf wird verzichtet, da die Antworten sowieso nicht gespeichert werden. Ein weiterer Grund, wieso auf das zweite Menu in diesem Fall verzichtet wird, ist, dass währenddem das Programm nicht läuft, nichts auf die SD-Karte geschrieben werden kann. Daraus folgt, dass die Systemvariabeln nicht gesetzt werden können und so auch die im Kapitel 4.4.2 beschriebenen vier Fälle nicht unterschieden werden können.

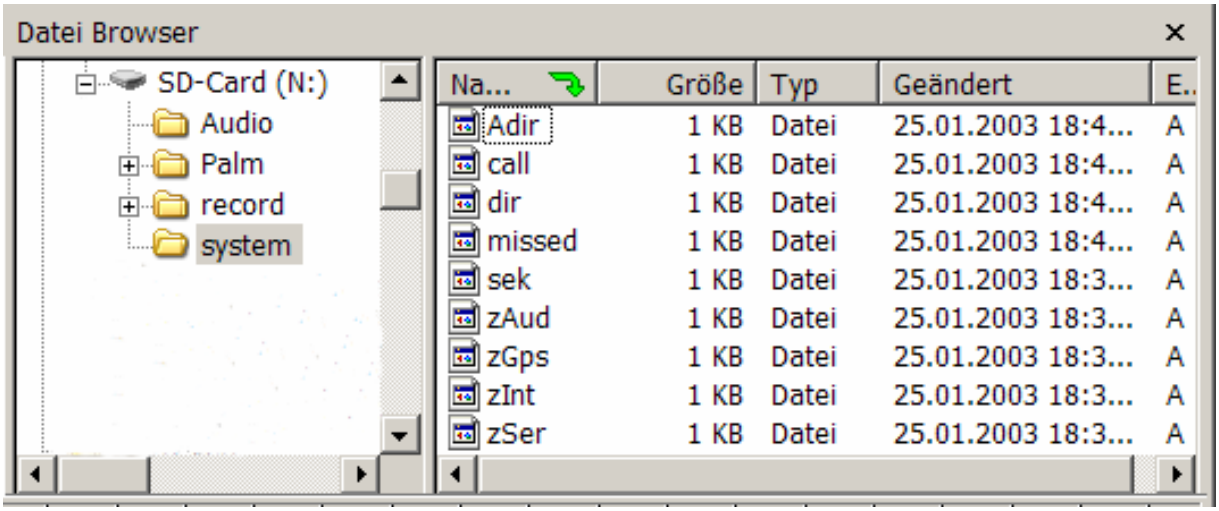
Um zu unterscheiden, ob das Programm gerade läuft oder nicht, kann das „LaunchFlag“ abgefragt werden. Dies lässt sich mit der folgenden Abfrage machen:

```
//wird nur ausgeführt, wenn Hauptprogramm läuft....
     if (launchFlags & sysAppLaunchFlagSubCall){
     }
//wird nur ausgeführt, wenn Programm nicht läuft
if (!(launchFlags & sysAppLaunchFlagSubCall)){
     }
```

4.4.4 Ablauf der Speicherung

Als letzter Teil des Ablaufes wird noch gezeigt wie und wann die Sensordaten und die Menueinträge abgespeichert werden.
In der Endlosschleife des Hauptprogramms werden die Sensorwerte aufgenommen und direkt in einen Ringbuffer abgefüllt. Sobald ein Anruf eingeht, wird die Zeit mittels eines Timestamps festgehalten. Dieser Timestamp und auch die Zähler der Ringbuffer werden jeweils in einzelne Dateien im Verzeichnis „System“ geschrieben, damit das Programm auf diese Werte auch aus anderen „Launchcodes“ wieder zugreifen kann. Anschliessend wird das erste Menu gestartet und sobald dieses beendet ist, wird der ausgefüllte Wert in dem Verzeichnis „temp“ abgespeichert, in welchem sich bereits die Audiodateien befinden. Falls der Anruf verpasst wird, wird dies ebenfalls in dieser Datei festgehalten. Sobald der Anruf beendet und das zweite Menu ausgefüllt ist, werden die Menueinträge und die Sensorwerte aus dem Ringbuffer in das Verzeichnis „temp“ gespeichert. Anschliessend wird das Verzeichnis „temp“ zu einem Verzeichnis mit dem Name des „Timestamps“ umbenannt.

In der folgenden Graphik ist der Inhalt des Verzeichnisses „system“ auf der SD-Karte zu sehen, in welchem alle Dateien gespeichert sind, welche dazu verwendet werden um auf Variabeln aus anderen „Launchcodes“ zugreifen zu können.



(Abb. 69: Dateien für Systemvariabeln)

Da das Programm ab und zu die Systemvariabeln auf die SD-Karte speichert und wieder abfragt[18], ist es unbedingt notwendig, dass, während dem das Programm lauft, einen SD-Karte eingesteckt ist. Wird die SD-Karte während dem Laufen des Programms entfernt kann dies zu einem Absturz führen. Um diesem Problem vorzubeugen, wird der Event, wenn die SD-Karte aus dem Slot entfehrnt wird, mit der „Notification“ „`sysNotifyCardRemovedEvent`“ abgefangen und eine selbst definierte Warnung auf dem Bildschirm angezeigt.

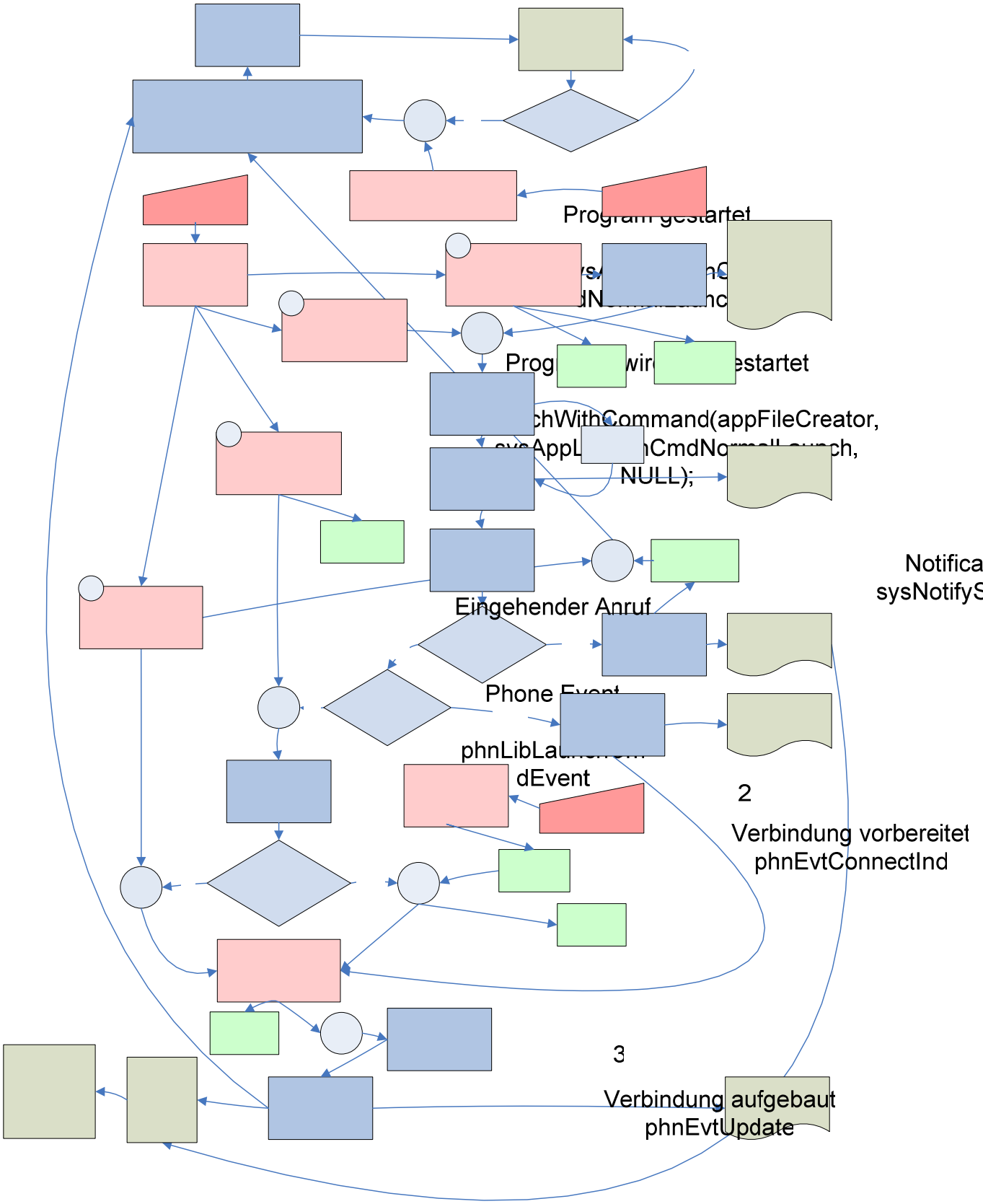
(Abb. 70: Fehlermeldung bei entfernter SD-Karte)

So bleibt dem Benutzer genügend Zeit die SD-Karte wieder einzustecken und nachher auf OK zu drücken.

4.4.5 Gesamter Ablauf des Programms

Als Abschluss des Kapitels 4.4 „Ablauf des Programms“ werden nun, in dem sich auf der nächsten Seite befindenden Diagramm, alle Einzelteile des Ablaufes des Programms zusammengefügt. Dieses Diagramm bietet eine schöne Übersicht über den gesamten Ablauf der Applikation.

[18] Durch den Audiostream wird ebenfalls dauernd auf die SD-Karte gespeichert.



(Abb. 71: Gesamter Ablauf des Programms)

4.5 Persistenz des Programms

Um die Applikation vor dem Benutzer zu verbergen ist es möglich, mit dem Freewareprogramm FileZ [nosleep,2006] das Programm zu verstecken. So sieht der Benutzer die Applikation nicht und kommt auch nicht auf die Idee sie zu löschen.

Damit unser Programm auch noch nach einem Hardreset vorhanden ist, gibt es die Möglichkeit ein eigenes ROM, welches die Applikation bereits enthält, zu erstellen und auf den Treo zu laden.
Um dies zu machen muss als erstes aus einem Softwareupdate von Palm [Palm,2006] das ROM extrahiert werden. Mit der neusten Softwareversion 1.20 welche auf der Palmseite [Palm,2006] zu finden ist, kann das ROM nicht extrahiert werden. Aber über diverse P2PSoftware können ältere Updates gefunden werden. Mittels des Freeware Java Programms T3 [Shadowmite,2006] lässt sich aus dem Update das ROM herausholen. Das dafür nötige Vorgehen wird auf der Seite [Shadowmite,2006], auf welcher sich das Javaprogramm herunterladen lässt, genau beschrieben. Auf dieser Seite findet sich zusätzlich das Programm T650Tool mit welchem dem ROM Programme hinzugefügt oder entfernt werden können.
Das Erstellen und Verwenden eines eigenen ROM's ist noch nicht vollständig überprüft und muss vor dem Einsetzen noch getestet werden.

Um sicherzustellen, dass alle weiteren Ordner, Dateien und Datenbanken, welche für die Applikation benötigt werden, vorhanden sind, wird bei Programmstart überprüft, ob diese existieren und falls dies nicht der Fall sein sollte, werden sie neu erstellt. So gefährdet es nicht den Experimentablauf, falls der Benützer aus Versehen etwas löscht.

5 Rahmenbedingungen für das Experiment

Durch die Implementierungen und Designentscheidungen, welche im letzten Kapitel beschrieben wurden, ist der Rahmen, in welchem das Experiment durchgeführt werden kann, entscheidend geprägt. Um die daraus entstehenden Rahmenbedingungen darstellen zu können, wird im Folgenden der Ablauf des Experimentes aus der Sicht eines Probanden geschildert.

Der Proband hat gerade seinen Treo erhalten und betrachtet das angeschlossene Sensorboard genauer. Den Bluetooth-GPS-Empfänger hat er pflichtbewusst eingeschaltet und in die offene Brusttasche seines Hemdes gesteckt. Da geht auch schon zum ersten Mal ein Anruf ein. Das Telefon stellt einen Fragebogen auf dem Bildschirm dar, auf welchem er antippen muss, wie stark er nun durch diesen Anruf gestört wurde. Dieses Menu ist mit einem einzigen Klick erledigt. Der Proband ist gezwungen den Anruf entgegenzunehmen. Sobald das Gespräch beendet ist, kommt ein zweiter Fragebogen auf das Display, in welchem er noch weitere Fragen beantworten muss.
Am Abend kommt der Proband nach Hause, stellt fest, dass der Treo ausgeschaltet ist und erinnert sich daran, dass ihm gesagt wurde, dass der Akku nur 8 h hält. Sogleich nimmt er den Zweitakku aus der Tasche und setzt ihn ein. Der andere Akku kommt in die Ladestation und schaltet seinen Treo wieder ein.

Um zu zeigen, was sich alles in diesem Beispiel abgespielt hat, wird das gleiche Beispiel nochmals, diesmal aber aus einer „high-level“ Sicht, erzählt.

Der Proband hat gerade seinen Treo, das Sensorboard und den GPS-Empfänger erhalten, betrachtet den Bildschirm des Treo's und sieht, dass das Programm zur Aufnahme läuft, da geht auch schon zum ersten Mal ein Anruf ein. Das Telefon deaktiviert sogleich die Tastensperre und stellt einen Fragebogen auf dem Bildschirm dar, auf welchem er antippen muss, wie stark er nun durch diesen Anruf gestört wurde. Währenddem dieser Fragebogen auf dem Display ist, vibriert das Telefon regelmässig und es ertönt ein Klingelton. Sobald das Menu ausgefüllt ist, schliesst es sich und der normale Bildschirm des Treo's, mit welchem man die Anrufe entgegen nehmen oder ablehnen kann, öffnet sich. Der Proband kann nun den Anruf nur mit der Hardwaretaste annehmen, da der Touchscreen und die Hardwaretaste zum Ablehnen deaktiviert sind. So kann er keinen Anruf ablehnen. Sobald das Gespräch beendet ist und einer von Beiden die Verbindung beendet hat, kommt ein zweiter Fragebogen auf das Display, in welchem er noch weitere Fragen beantworten muss. Sobald dieser Fragebogen ausgefüllt ist, erscheint ein Hinweis, dass nun die Sensordaten gespeichert werden. Dies kann bis zu 30 Sekunden dauern. Ist diese Zeit abgelaufen, startet automatisch wieder das Programm zur Sensoraufnahme.
Da er in eine wichtige Vorlesung muss, schaltet er, trotz den Anweisungen des Experiment-Leiters, den Ton aus und legt das Telefon in den Rucksack. Kurz darauf kommt ein zweiter Anruf, das Programm aktiviert das Display, die Tastenbeleuchtung und startet wieder das erste Menu. Er hört und sieht aber nichts davon und verpasst prompt diesen Anruf. Das Telefon hat in der Zwischenzeit den ersten Fragebogen wieder geschlossen, die Treo eigene Telefonapplikation registriert, dass ein Anruf verpasst wurde und bringt eine Nachricht mit der Nummer des Anrufers auf den Bildschirm. Ebenfalls werden hier alle Sensordaten mit dem Vermerk, dass der Anruf verpasst wurde, auf der SD-Karte gespeichert. Das zweite Menu wird nicht mehr gestartet. Nach 30 Sekunden wird das Programm zur Sensoraufnahme wieder gestartet

und die Nachricht verschwindet. Am Abend kommt der Proband nach Hause, stellt fest, dass der Treo ausgeschaltet ist und erinnert sich daran, dass ihm gesagt wurde, dass der Akku nur 8 h hält. Sogleich nimmt er den Zweitakku aus der Tasche und setzt ihn ein. Der andere Akku kommt in die Ladestation. Sobald der Treo nun wieder eingeschaltet ist, wird er auch sofort aufgefordert seinen Pincode einzugeben und die Telefonfunktion des Treo's wird aktiviert. Um die Daten auf der SD-Karte muss er sich nicht kümmern, er weiss nur, dass er in einem Monat wieder vorbeikommen und die SD-Karte gegen eine leere austauschen muss.

6 Zusammenfassung

In dieser Arbeit wurden die Fragmente meiner Vorgänger zu einem gesamten Programm zusammengefügt und ergänzt, so dass es den Ansprüchen genügt, welche benötigt werden um das Experiment durchführen zu können.

So wurde das Menu von Manuel Donner, welches eine reine Designarbeit war, verfeinert, ergänzt und mit Funktionalität ausgestattet (z.B. Abspeicherung der Antworten).

Der Sensorrecorder von Robin Bucciarelli wurde so verändert, dass bei dem Programmstart automatisch mit den Aufnahmen begonnen wird. Das Programm wurde um eine Funktion ergänzt, mit welcher kontinuierlich die internen Variabeln des Treo's ausgelesen werden. Zusätzlich wurde ein Ansatz geliefert, mit welchem evtl. das Problem der Aufnahme der Daten des Sensorboardes behoben werden kann.

Um die von den Sensoren erhaltenen Daten abspeichern zu können, wurde ein Ringbuffer mittels einer Datenbank und mittels Dateien konstruiert. Damit diese abgespeicherten Daten geordnet und einem Event zu gewiesen werden können, wird fortlaufend eine Ordnerstruktur erstellt.

Der wohl aufwändigste Punkt war die Abhandlung des gesamten Ablaufs.
Es wird sichergestellt, dass das Programm zur Sensoraufnahme immer läuft und falls es vom Benutzer beendet wird, wieder neu gestartet wird, sobald das Telefon nicht mehr für etwas anderes gebraucht wird.
Das Abfangen eines Anrufs und das Dazwischenhängen eines Fragebogens vor und nach dem Anruf wurde für alle möglichen Fälle erfolgreich abgehandelt. Nach dem Anruf werden alle Sensordaten auf die SD-Karte gespeichert.

6.1 Limitierungen

Die grösste Limitierung des Programms ist es, dass die Aufnahme der Daten des Sensorboardes erst ansatzweise funktioniert. Es ist sehr schwierig abzuschätzen, wie viel Zeit benötigt wird um eine Parameterkombination zu finden, welche akzeptable Resultate liefert.

Sobald die Aufnahme der Sensordaten funktioniert, muss das Programm noch um eine Funktion erweitert werden, welche nach dem Beenden des Anrufes die Gassensoren aktiviert und eine kurze Aufnahme macht, solange der Benutzer den zweiten Fragebogen ausfüllt. Für diesen Zweck muss lediglich die Funktion zum Öffnen und Schliessen des seriellen Ports und das Senden und Empfangen der Daten in den Programmteil des zweiten Menus immigriert werden. Dies ist sicherlich problemlos machbar, es muss lediglich beachtet werden, dass unter einem anderen Launchcode die globalen Variabeln nicht mehr verfügbar sind.

Eine weitere Einschränkung ist, dass die Funktion, welche in konstanten Abständen im Hintergrund mit der integrierten Kamera Photos machen soll, bis jetzt noch nicht funktioniert. Das Email von Szymon Ulatowski [Ulatowski, 2006] bietet hierfür eine gute Grundlage. Sofern es sich nicht um einen Fehler oder eine Einschränkung in dem Betriebssystem des Treo's handelt, sollte es relativ einfach umsetzbar sein.

6.2 Anmerkungen und Ausblick

Es wäre sehr spannend gewesen, das Experiment durchzuführen und zu begleiten. Gerne hätte ich die erhaltenen Daten ausgewertet und analysiert. Doch durch die vielen unvorhersehbaren Probleme und Einschränkungen in der Umsetzung des Programms für dieses Experiment, ist diese Arbeit eine reine Evaluations- und Implementierungsarbeit geworden.

Ein weiterer Schritt zur Entwicklung eines Anruffilters wäre sicher auch, wie in der Einleitung bereits beschrieben, der Einsatz einer Spracherkennungssoftware, welche selbständig Kategorien anlegt, neue Inhalte diesen Kategorien zuordnet und so einen weiteren, sehr wichtigen Anhaltspunkt für die Klassifizierung der Unterbrechbarkeit liefert.
Sehr interessant wäre es, wenn zusätzlich noch Biosensoren[19] an der Testperson angebracht wären, wie Körpertemperatur und Herzschlag. Diese Sensoren würden eine sehr gute Vorhersage für die Unterbrechbarkeit des Benutzers zeigen, da emotionale Zustände wie Stress oder Anspannung sich sofort in den Biowerten widerspiegeln.
Bei einer weiterhin konstanten Entwicklungsgeschwindigkeit der mobilen Kommunikationsgeräte ist es sicher bald möglich, das gesamte „preprocessing“ und die Klassifikation direkt auf dem mobilen Gerät durchzuführen und so diesen Filter umzusetzen.

[19] Biosensoren wurden schon im Zusammenhang mit „Context awareness“ verwendet. Aber explizit die Unterbrechbarkeit wurde meines Wissens nach noch nie mit Biosensoren getestet.

7 Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, welche mich bei der Erstellung dieser Diplomarbeit unterstützt haben.

Für die gute und ausführliche Beratung zur Themenwahl möchte ich mich ganz herzlich bei Prof. A. Bernstein bedanken.

Ein ganz grosses Dankeschön geht auch an Peter Vorburger, der mich immer sehr gut unterstützt und mich durch anregende Gespräche wieder auf neue Lösungsansätze gebracht hat.

Zusätzlich bedanken möchte ich mich auch bei allen Personen, welche meine Arbeit zur Korrektur gelesen haben.

Bedanken möchte ich mich auch bei meinen Eltern, welche mir durch ihre Unterstützung erst dieses Studium ermöglicht haben.

8 Verzeichnisse

8.1 Abbildungsverzeichnis

8.2 Literaturverzeichnis:

[Abraham Bernstein, 2005] Abraham Bernstein, P. V. (2005). "A Scenario-Based Approach for Direct Interruptablity Prediction on Wearable Devices." Department of Informatics, University of Zurich.

[Abraham Bernstein 2, 2005] Abraham Bernstein, P. V., Patrice Egger (2005). "Direct Interruptablity Prediction and Scenario-based Evaluation of Wearable Devices: Towards Reliable Interruptability Predictions." Department of Informatics, University of Zurich,.

[Bachmann, 2006] Bachmann, A. (2006). "Indoornavigation mittels Ortsinterpolation." Master's thesis in Computer Science, University Zürich.

[Brian, 1988] Brian W.Kernighan, D. M. R. (1988). "C Programming Language." Prentice Hall PTR.

[Bucciarelli, 2006] Bucciarelli, R. (2006). "Enabling a mobile phone to sense its surroundings Integration of internal and external sensors in a mobile phone." Master's thesis in Computer Science, University Zürich.

[Conklinsystems, 2006] Conklinsystems (2006). "CS Online."

[Dey], Dey, "Towards a Better Understanding of Context and Context-Awareness."

[Donner, 2006] Donner, M. (2006). "Untersuchung zur Heterogenität von Benutzergruppen in mobilen, kontextbezogenen Anwendungen." Master's thesis in Computer Science, University Zürich.

[Fornallaz, 2004] Fornallaz, J. (2004). "Fundamental Implementation for Context Sampling on Mobile Phones." Master's thesis in Computer Science, University Zürich.

[Frank,, 2005] Frank, I. H. W. E. (2005). "Data Mining, Practical Machine Learning Tools and Techniques." Elsevier.

[Honeywell, 2006] Honeywell (2006), Datasheet HMC1053,
From: http://www.ssec.honeywell.com/
magnetic/datasheets/HMC1053.pdf
(Besucht am 27.4.06)

[Kotz] Kotz, G.. "A Survey of Context-Aware Mobile Computing Research."

[Lonnon, 2005] Lonnon R.Foster, G. B. (2005). "Professional Palm OS® Programming." Wiley Publishing, Inc.

[McKeehan, 1998] McKeehan, N. R. J. (1998). "Palm Programming: The Developer's Guide." O'Reilly.

[nosleep,2006] nosleep. (2006). "FileZ." from http://www.nosleep.net/. (Besucht am 27.4.06)

[Palm, 2006] Palm. (2006). "Palm Inc. Produkte", from http://euro.palm.com/ch/de/products/treo650/index.html. (Besucht am 27.4.06)

[PalmSource, 2006] PalmSource. (2006). "Palm Source Developers." from http://www.palmsource.com/developers/. (Besucht am 27.4.06)

[Preece, 2002] Jennifer Preece, Y. R., Helen Sharp (2002). "Interaction Design - beyond human-computer interaction." John Wiley and Sons Ltd

[Vorburger, 2005] Peter Vorburger, A. B. (2005). "Towards an Artificial Receptionist: Anticipating a Persons Phone Behavior." Department of Informatics, University of Zurich.

[Vorburger 2, 2005] Peter Vorburger, A. B., Alen Zurfluh (2005). "Preventing Efficiency by Reducing Interruptions - The Power of Multi-Sector Motion DetectionThe Artificial Receptionist: Anticipating a PersonsPhone Behavior." Department of Informatics, University of Zurich,.

[Roth, 2002] Roth, J. (2002). "Mobile Computing Grundlagen, Technik, Konzepte." dpunkt Verlag.

[Shadowmite,2006] Treo 650 Custom ROMs http://shadowmite.com/roms.html (Besucht am 27.4.06)

[Shneiderman., 1997] Shneiderman, B. (1997). "Designing the User Interface - strategies for effective Human-.Computer Interaction, third Edition." Addison Wesley

[Simoni, 2005] Simoni, F. d. (2005). "Auf dem Weg zum wirklich smarten Smartphone: Anbindung interner und externer Sensorik zur Kontexterfassung." Master's thesis in Computer Science, University Zürich.

[Stallings, 2003] Stallings, W. (2003). "Betriebssysteme Prinzipien und Umsetztung." Prentice Hall Pearson Studium.

[Stroustrup., 2000] Stroustrup, B. (2000). "Die C++ Programmiersprache." Addison-Wesley.

[Ulatowski, 2006] Ulatowski, S. (2006). "Video Jigsaw." from http://www.toyspring.com. (Besucht am 27.4.06)

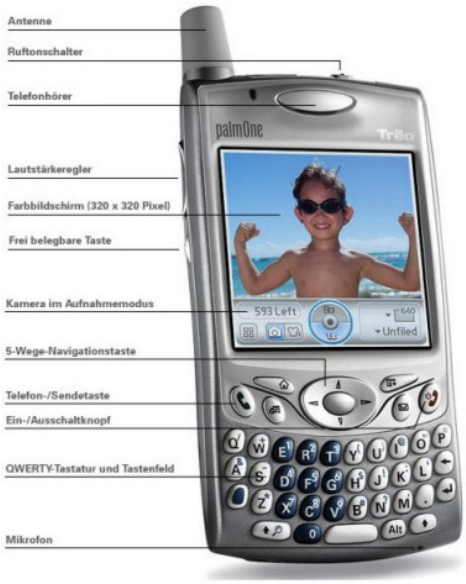
[Wikipedia, 2006] Wikipedia. (2006). from http://de.wikipedia.org/wiki/Hauptseite. (Besucht am 27.4.06)

[Willms, 1997] Willms, G. (1997). "C Das Grundlagenbuch." Data Becker.

[Zurfluh,, 2004] Zurfluh, A. (2004). "The Artificial Secretary Voraussage des Telefonierverhaltens in einem Bureau anhand von Sensordaten." Master’s thesis in Computer Science, University Zürich.

9. Anhang

9.1 Daten des Treo 650



- Quadband 850/900/1800/1900 MHz
- Abmessungen: 11,3 x 5,9 x 2,3 cm
- Gewicht: 178 Gramm
- Akku: Austauschbarer, wiederaufladbarer Lithium-Ionen-Akku
- Standby-Zeit: bis 300 Stunden
- Sprechzeit: 360 Minuten
- TFT-Touchscreen Display mit Handschriftfunktion
- Auflösung: 320 x 320
- Anzahl Farben: Über 65.000 Farben (16 Bit)
- Palm OS 5.4
- VGA Digital Camera 640x480 (0.3 Megapixel) automatischer Helligkeitsregelung
- Beleuchtete QWERTY-Tastatur
- Polyphone Klingeltöne
- Prozessor: Intel PXA270 mit 312 MHz
- Infrarot / Bluetooth
- SMS / MMS
- E-mail / Internet / Kalender / Agenda
- Vibra Alarm
- Speicher: 32 MB, 21 MB verfügbarer, nicht flüchtiger Speicher
- Erweiterung: Unterstützt SD-, SDIO- und MultiMediaCard-Erweiterungskarten

9.2 Notificationen:

In der folgenden Tabelle sind alle „Notificationen“ und deren Verwendungszweck aufgelistet:

Constant	Description
cncNotifyProfileEvent	The connection profile used by the Connection Panel has changed.
kTelTelephonyNotification	A telephony event has occurred.
Socket Notification	A network socket change has occurred.
sysExternalConnectorAttachEvent	A device has been attached to an external connector.
sysExternalConnectorDetachEvent	A device has been detached from an external connector.
sysNotifyAntennaRaisedEvent	The antenna has been raised on a Palm VII™ series device.
sysNotifyAppLaunchingEvent	An application is about to be launched.
sysNotifyAppQuittingEvent	An application has just quit.
sysNotifyCardInsertedEvent	An expansion card has been inserted into the expansion slot.
sysNotifyCardRemovedEvent	An expansion card has been removed from the expansion slot.
sysNotifyDBCreatedEvent	A database has been created.
sysNotifyDBChangedEvent	Database info has been set on a database, such as with DmSetDatabaseInfo().
sysNotifyDBDeletedEvent	A database has been deleted.
sysNotifyDBDirtyEvent	A database has been opened for write or in some other way has been made modifiable.
sysNotifyDeleteProtectedEvent	The Launcher has attempted to delete a protected database.
sysNotifyDeviceUnlocked	The user has unlocked the device.
sysNotifyDisplayChangeEvent	The color table or bit depth has changed.
sysNotifyDisplayResizedEvent	The dynamic input area has opened or closed.
sysNotifyEarlyWakeupEvent	The system is starting to wake up.
sysNotifyEventDequeuedEvent	An event has been removed from the event queue with EvtGetEvent.

sysNotifyForgotPasswordEvent	The user has tapped the Lost Password button in the Security application.
sysNotifyGotUsersAttention	The Attention Manager has informed the user of an event.
sysNotifyGsiDrawIndicator	The system is about to draw the shift indicator.
sysNotifyHelperEvent	An application has requested that a particular service be performed.
sysNotifyIdleTimeEvent	The system is idle and is about to doze.
sysNotifyInputAreaDrawingEvent	The system is about to draw the dynamic input area.
sysNotifyInputAreaPendownEvent	The system is about to post a penDownEvent for the dynamic input area.
sysNotifyInsPtEnableEvent	The insertion point is being enabled or disabled.
sysNotifyIrDASniffEvent	Not used.
sysNotifyKeyboardDialogEvent	The keyboard dialog is about to be displayed.
sysNotifyLateWakeupEvent	The system has finished waking up.
sysNotifyLocaleChangedEvent	The system locale has changed.
sysNotifyMenuCmdBarOpenEvent	The system is about to display the menu command toolbar.
sysNotifyNetLibIFMediaEvent	The system has been connected to or disconnected from the network.
sysNotifyPhoneEvent	Reserved for future use.
sysNotifyPOSEMountEvent	System use only.
sysNotifyProcessPenStrokeEvent	The user has made a pen stroke on the silkscreen portion of the digitizer.
sysNotifyResetFinishedEvent	The system has finished a reset.
sysNotifyRetryEnqueueKey	The Attention Manager has failed to post a virtual character to the key queue.
sysNotifySelectDay	The system needs to request that the user pick a particular date in the calendar.
sysNotifySleepNotifyEvent	The system is about to go to sleep.
sysNotifySleepRequestEvent	The system has decided to go to sleep.
sysNotifySyncFinishEvent	A HotSync® operation has just completed.
sysNotifySyncStartEvent	A HotSync operation is about to begin.

sysNotifyTimeChangeEvent	The system time has just changed.
sysNotifyVirtualCharHandlingEvent	A virtual character is being handled.
sysNotifyVolumeMountedEvent	A file system has been mounted.
sysNotifyVolumeUnmountedEvent	A file system has been unmounted.

9.3 PhoneEventCodes

In der folgenden Liste sind alle Parameter des Launchcode, welcher bei einem Phoneevent abgefangen werden können.

phnEvtCardInsertion	0x0000 < NOT USED. OBSOLETE
phnEvtRegistration	Network Registration has changed.
phnEvtError	Error detected
phnEvtKeyPress .	phone-related keypress has occured on device
phnEvtPower	change in power state of radio
phnEvtPassword	Password Dialog Control
phnEvtProgress	outgoing call progress dialog
phnEvtIndication	indication of various changes of state in Phone Library
phnEvtConnectInd	incoming call received
phnEvtConnectConf	call has connected
phnEvtSubscriber	Number/name of call has changed
phnEvtDisconnectInd	call has disconnected
phnEvtDisconnectConf	call in conference (or 3-way calling) mode has disconnected
phnEvtBusy	Outbound call attempt failed because remote party (or network) reported busy
phnEvtUpdate	Change in Call Status; Application may need to update its state
phnEvtConference	This notification occurs when a call goes into conferrence (or 3-way calling) mode.
phnEvtVoiceMail	Incoming VoiceMail Message
phnEvtMessageInd	Incoming SMS message received
phnEvtSegmentInd	Incoming segment of SMS Message received
phnEvtMessageStat	Status of outgoing SMS message has changed
phnEvtMessageDel	SMS message deleted from local DB on device
phnEvtMessageMoved	SIM Contains SMS Messages
phnEvtSATNotification	SIM Application Toolkit (SAT) event
phnEvtUSSDInd	USSD input/output requested from network
phnEvtPhoneEquipmentMode	Phone Equipment state has changed
phnEvtGPRSRegistration	GPRS Attach Registration has changed. (GSM/GPRS systems only)
phnEvtMMSInd	incoming MMS message available
phnEvtMemoryFull	SMS Memory Full

phnEvtMemoryOK	SMS Memory no longer full
phnEvtWAPInd	WAP Message received via SMS (WAP in IS-637 standard)
phnEvtDebugReport	Radio Debug Info
phnEvtSSModeChanged	System Selection mode has changed
phnEvtPDDataChanged	Position Info has changed
phnEvtOneXStatus	1XRTT Status has changed
phnEvtMIPFailed	Mobile IP connection failed
phnEvtIOTAStatus	IOTA Status has changed
phnEvtDataChannel	Data Channel availability has changed
phnEvtOneXMipRRQInd	Mobile IP Registration successful
phnEvtOneXDataFail	1XRTT Session has failed (mid-session)
phnEvtStartDial	Start Dialing of Outbound Call
phnEvtStartIncomingCall	Start of incoming call
phnEvtAlertingPreConnected	Alerting/Preconnected of Outbound Call
phnEvtAlerting	Alerting of Outbound Call (NOT USED, for future implementation)
phnEvtPreConnected	Preconnected of Outbound Call (NOT USED, for future implementation)
kMaxPhnEvtSupported	Must be at last position

9.4 Liste der Requirements

Um alle Anforderungen an das Programm richtig umsetzen zu können, habe ich fortlaufend eine Liste geführt, in welcher ich alle Ansprüche an das Programm und die möglichen Situation aufgelistet, überprüft und abgearbeitet habe.

GUI Tasks:

- Screen zur Bewertung der Entscheidung -> Kostenfunktion->ROC-kurve; OK
  - Wie wichtig oder egal ist es, dass der Anruf durchgestellt oder abgelehnt worden ist
  - Slidebalken (ROC) überarbeiten und Wert zuordnen
- Auflösung verfeinern; OK
- Notifikationscreen: Icons neu gestallten
  - Systemmessage, Systembeep, Vibra; OK

Programmier Tasks:

- Sicherstellen, dass Programm immer läuft oder wieder gestartet wird.
  - Launchcodes, Notifications; OK
- Ringbuffer für Messdaten; OK
- Anrufzwischenschaltung: GUI, Messdaten speichern; OK
- Annotationsdaten speichern; OK
- Ordnerstruktur für Mess- und Annotationsdaten; OK

Allg. Tasks:

- ROM brennen, damit das Program einen HardReset übersteht
- Mit Freeware Programm „Filez“ Programm vor Anwender verstecken. OK
- Evaluation: Telephonabnahme Möglichkeiten; OK
- Sind 2GB SDCards vom Treo unterstützt? OK
- **Datenmenge Vermessung:** OK
  - Wie gross sind die Daten
  - Wie grosse Daten werden von Phone bzw. SD-Karte noch verarbeitet
  - wird regelmässig aufgenommen
  - Wird auch während „standby“ aufgenommen

Programmieren:

Programm Ablauf

- Programm für Notifications registrieren…wird bei Synchronisation oder Softrest durch geführt; funktioniert

- Bei Bildschirausschaltung (sysNotifySleepNotifyEvent) wird das Programm neu gestartet, falls es nicht schon läuft; funktioniert
  Benütze „sysNotifySleepRequestEvent“ damit Programm startet, bevor „sleepmodus“ eintrifft

```
if (!(launchFlags & sysAppLaunchFlagSubCall)) {
AppLaunchWithCommand(appFileCreator, sysAppLaunchCmdNormalLaunch,
NULL);
}
```

- Beim Neustart des Programms den Display aktivieren, damit das Recording beginnt; funktioniert
- Bei Programmstart erst Serial, dann Bluetooth und zuletzt Audio-recording automatisch starten; funktioniert
  Dieser Ablauf befindet sich in der Funktion SERMainFormHandleEvent
- Bei Programm Beendung erst Serial, dann Bluetooth und zuletzt Audio stoppen; funktioniert
  befindet sich in der Funktion SERMainFormHandleEvent

Anrufzwischenschaltung

- Notification für Telephonanruf existiert nicht (kTelTelephonyNotification) PalmSource – PalmOne

- Für Phoneevents registrieren; funktioniert
  PhnLibRegister(PhoneLibRefNum, 'TRER', phnServiceVoice);
- Phone Event abfangen (phnLibLaunchCmdEvent); funktioniert
- Event unterscheiden und einkommenden Anruf abfangen; funktioniert
  (phoneEvent->eventType == phnEvtStartIncomingCall)

- Rufton und Lautstärke fixieren ist; funktioniert
  HsPrefSet
- Bildschirm wieder einschalten; funktioniert
- Tastensperre wieder aufheben und Tastatur beleuchten; funktioniert
- Hardware Butten unterdrücken mit HsKeyEnableKey; funktioniert
  Namen und Hex der Tasten findet sich in der Libary Datei HsKeyCodes.h
- Menu PersStoerForm starten; funktioniert
- Rington während dem Menu; funktioniert
- Vibrieren während dem Menu; funktioniert
- Counter eingeführt, dass erstes Menu nach 12 sek sich selbst schliesst, damit bei einem verpassten Anruf die eigene Telefonapplikation übernehmen kann; funktioniert

- Nach Menu Telephone annehmen; funktioniert bis jetzt nicht, aber nicht nötig
- Befehl TelSpcAcceptCall(); nur in Palm Source nicht aber in PalmOne; fuktioniert nicht nicht nötig
- Tastendruck zum Abnehmen senden, EvtEnqueueKey(0x160A bzw. 0x0204); funktioniert nicht nicht nötig

- Menu schliessen, damit die Telefoneapplikation startet und der Benutzter selbst annehmen oder abweisen kann; funktioniert

- Unterdrückung der Hardware Abweisungstaste mit HsKeyEnableKey(); funktioniert
- Unterdrückung der Bildschirm Abweisungstaste durch Deaktivierung des Touchscreens; funktioniert
- Der Fall dass der Anruf verpasst wird abfangen; funktioniert
- Missed Call bis jetzt Absturz / Variable Missed
  sysNotifyDBChangedEvent -> notifiyDetailsP -> creator = ‘psys’

- Der Fall das der Benützer des Telefonat abweist ist; funktioniert
  Rejected Call

- Alternative: Menu erst nach dem Telefonat einbinden; nicht mehr nötig

- Verbindungsabbruch abfangen (phoneEvent->eventType == phnEvtDisconnectInd) wird nur gesendet wenn Gegenseite abhängt; funktioniert

- Der Fall, dass Benutzer abhängt; funktioniert
  Terminated Call by user
  sysNotifyDBChangedEvent creator = psys
  & Variabel Call

- Restliches bzw. ganzes Menu starten; funktioniert
- Hardware Butten unterdrücken mit HsKeyEnableKey; funktioniert
- Time-Out für Annotation einsetzten; nicht nötig -> Designfrage

- Applikation neu starten; funktioniert

Abspeichern

- 4 Ringbuffer für verschiedene Messdaten; funktioniert
  Sensorboard, Audio, GPS, Internal and maybe Photo
  Buffer in Memory nicht möglich (68K grenze).
  Workaround mittels DB

- Interne Sensoren aufnehmen funktioniert
  Tastensperre, Batteriestärke, Sendestärke, mute oder laut, evtl. Sendezelle und Anrufername
- Timestamp (Anzahl Sekunden) um Recorddauer genau zu haben (max 600 sec) funktioniert

Bei Programm start

- Im Ordner „Record“, „temp“ Ordner anlegen funktioniert
  Um bei Beendung des Programms die Daten abzulegen
- In dem Ordner die Dateien „Serial.txt“, „gps,txt“ & „audio“ erstellen funktioniert
  Um Sensordaten welche bei Beendung des Programms gespeichert werden müssen zu speichern.
- Datei „dir“ enthält Name des Ordners in welchem die Sensordaten abgespeichert werden sollen / dir = temp funktioniert

Bei Incoming call -> (phoneEvent->eventType == phnEvtStartIncomingCall):

- Im Ordner „Record“ einen Ordner mit Timestamp erstellen funktioniert
- In dem Ordner die Dateien Serial.txt, gps,txt & audio erstellen funktioniert
- Dir = timestamp funktioniert

Bei Stop Application:

- Buffer in diese Dateien speichern funktioniert

Bei call Vorbereitung -> (phoneEvent->eventType == phnEvtConnectInd):

- Nachdem das Menu PersStoerForm beendet -> 1. Annotation in Recordordner abspeichern funktioniert
- Falls phnEvtUpdate nicht erreicht wird -> missed call annotieren funktioniert

- Anzahl aufgenommene Sekunden in Ordner Timestamp in Datei „seconds.txt“ abspeichern funktioniert

Bei call Terminierung -> (phoneEvent->eventType == phnEvtDisconnectInd):

- Nach restlichem Menu 2. Annotation in Record abspeichern funktioniert
- Gassensoren messen und in Temp abspeichern noch zu machen

Bei Programm Neustart:
- Dateien im Ordner Temp leeren funktioniert
- Menuscreen „bitte warten Sensordaten werden gespeichert“ funktioniert

Buffer:

- DB's als Buffer missbraucht, da für Programme nur ca. 40kb Memory zu Verfügung stehen
- DB's falls noch nicht vorhanden erstellen und öffnen funktioniert
- 3 Datenbanken erstellt für Internal, Serial und GPS Sensor inputs als Ringbuffer funktioniert
- Interne Sensorwerte alle zu einem String vereint (strcat) funktioniert
    - nun wird zu schnell aufgenommen →Fehler: (GSM! Bei zu ofter Signalstärke abfrage)
- Counter zur Ausbremsung einbauen (mittels SysTicksPerSeconds(); damit Messwerte immer genau einmal pro Sekunde eintreffen) nun do mit TimGetSeconds(); -→ eine Messung pro Sekunde funktioniert
- Maximale Grösse der DB's festlegen damit genau 10min Sensordaten darin gespeichert werden können funktioniert
- Sensordaten String in DB abspeichern (Zähler = ID des DB-Eintrages) funktioniert
- Bei beenden des Programms → alle DB-Einträge in richtiger Reihenfolge auslesen und im richtigen Ordner in Datei speichern. funktioniert
- DB's schliessen, löschen und neu erstellen. Funktioniert
- DB's bei Programmstart löschen und wiedererstellen, da sonst sie evtl. nicht gelöscht werden. Und so wird der Speicherplatz belegt, so das der Benutzer nicht Programme installiert und den Platz belegt Funktioniert
- DB's nach Neustart des Treo's löschen und wiedererstellen, für den Fall, dass während dem Telefonieren der Akku leer ist. Funktioniert

- Intern = 1 Eintrag pro Sekund –Max = 600 , Serial = 20 Einträge pro Sek – Max = 12'000, GPS = 6.5 einträge pro Sek – Max = 3900…
- Intern Zeile = 74 Zeichen, Serial Zeile = ca. 50 Zeichen, GPS Zeile =
- die Sensordaten normal wie bisher auf die Sd-Karte gespeichert und zusätzlich über den RingBuffer abgespeichert und beide Dateien verglichen → richtige Werte

- Zähler der Ringbuffer in Datei gespeichert und in Funktion SaveSensordata wieder auslesen → somit kann nun überall die Funktion SaveSensordata aufgerufen werden

Audio

- Alle 2min neu Datei beginnen nach 10min die erste wieder überschreiben… funktioniert
- Bei frmclose in Datei speichern welche Datei die erste ist funktioniert
- In Funktion savesensordata audio/temp in audio/timestamp umbenennen funktioniert

Den Namen der ersten Audiodatei in Datei „reihenfolge.txt" der erste Dateiname speichern funktioniert

- Abspeicherung:
    - Bei Programm beenden und (Call = false) → nur DBreset funktioniert
    - Bei (phoneEvent->eventType == phnEvtDisconnectInd) und (missed = true) → SaveSensordata();funktioniert
    - Bei (cmd == contextRecorderLaunchCmdPhoneDisconnected) und (Call = true) → SaveSensordata();funktioniert

  Somit werden nun auch die Sensordaten gespeichert auch wenn der Anruf verpasst wird funktioniert…

System DB

- Datenbank für globale Variabeln zu speichern, da nach eingehendem Anruf kein Zugriff auf globale Variabeln möglich ist
- 1 Eintrag Call (true = unabgeabeiterter anruf, false = anruf erledigt)
  2 Eintrag missed (true = anruf verpasst, false = anruf angenommen)
  3 Eintrag Verzeichnis record/timestamp um Sensordaten zu speichern
  4 Eintrag Verzeichnis audio/timestamp um audiodaten zu speichern
  5 Eintrag wie viele Sekunden aufgenommen wurde
  6 Eintrag Welche Audiodatei (a,b,c,d,e)
  7 Eintrag Zähler des Internal Ringbuffers
  8 Eintrag Zähler des GPS Ringbuffers
  9 Eintrag Zähler des Serial Ringbuffers

Aus Stabilitätsgründen anders gelöst: pro globale Variable eine Datei, in welche geschrieben und wieder ausgelesen wird.

Sonstiges

- Telefon nach Reset wieder aktivieren
- Telefon bei Programmstart einschalten falls es ausgeschaltet war, jetzt ohne Bestätigung. Funktioniert

- Programm um gedrückte Tasten anzuzeigen, um Telefontaste herauszufinden ~funktioniert

- Integration von Robins Programm funktioniert
- Sensoraufnahmen automatisch bei Programmstart anfangen und bei Beendung des Programms stoppen. Funktioniert
-
- Existenz der Dateien auf der SD-Karte überprüfen, falls nicht vorhanden -> erstellen funktioniert
- Aufsplittung des Programms in Module noch zu machen

- Close Serial Port bevor Sync da sonst der Port schon belegt und sync nicht möglich Funktioniert
- Datei verstecken, damit User sie nicht findet… funktioniert
- Eigenes Rom erstellen -> damit Programm Hardreset übersteht noch zu machen

Bugs

- Wenn bei einem einkommenden Anruf auf „Ignor“ gedrückt wird -> weisser Bildschirm behoben
- Menupunkt Anrufer Name -> Absturz nach OK bei „not filled in“ behoben
  Lösung: mittels custombased notification das Menu in einem anderen Kontext ablaufen lassen.
- Menupunkt Anrufer Name -> Absturz nach OK bei „not filled in“ behoben
- Absturz bei Beendung des Programms behoben
  Bluetooth sauber beenden

- Programm beginnt nicht aufzunehmen wenn es durch Standby Notification gestartet wird behoben
  Undokumentierter Notificatio (sysNotifySleepRequestEvent) gefunden -> möglich Standby zu verzögern
- Falls Anruf erfolgt, während das Programm nicht lauft kommt nach Verbindungsaufbau Absturz behoben
  Phone Notifications werden nun nur abgehandelt, wenn Sensorprogamm gelaufen ist…
  Nachteil: Verschiedene Sichten für Benutzer

- Verpasster Anruf -> Absturz behoben
  Mögliche Lösung: Timeout -> Menu PersStoerForm beenden -> Phoneapplication handelt fall Call missed (löst Problem nicht).

- Falls während dem zweiten Menu die SD-Karte entfernt und wieder eingesetzt wird, wird das Menu übersprungen und direkt das Programm wieder gestartet
- Programmstart ohne SD-Karte - > Absturz

9.5 Email von Szymon Ulatowski

```
in my other application i use kCamLibCtrlCapture to get a single frame
with higher resolution and this way is probably better if you don't need
video.
assuming these two helper functions and "camref" being the camera library
reference:

#define CAMDO(control,param) CamLibControl(camref,control,param)

long setCam(long opt,long val)
{
CamLibSettingType settingType; settingType.type = val; return
CamLibControl(camref, opt, &settingType); }

you can capture the single frame this way:
    CamLibCaptureType capparams={&camlibCaptureFun,0};

setCam(kCamLibCtrlCaptureFormatSet,kCamLibCaptureDataFormatYCbCr422); //
or kCamLibCaptureDataFormatRGB565
    setCam(kCamLibCtrlCaptureSizeSet,kCamLibImageSizeVGA);
    CAMDO(kCamLibCtrlCapture,&capparams);

(don't ignore error codes in the real application!)

the image will be passed to your another function (referenced in
capparams) here i'm saving the raw image data to the storage file stream:

Err camlibCaptureFun(void *bufP, UInt32 size, void *userDataP) { FileHand
f; if (!size) return 0;
f=FileOpen(0,TMPFILENAME,'DATA',CRID,fileModeReadWrite,&err);
if (f) {FileWrite(f,bufP,1,size,&err);FileClose(f);}
return 0;
}

for SD card you would use VFSFileOpen/VFSWrite

eg.:

UInt32 it; UInt16 vol; Err err;
it=vfsIteratorStart;
if (!VFSVolumeEnumerate(&vol,&it))
    {

err=VFSFileOpen(vol,"/dump.bin",vfsModeReadWrite|vfsModeCreate|vfsModeTru
ncate,&fref);
    if (!err)
        {
        long done;
        VFSFileWrite(fref,size,bufP,&done);
        VFSFileClose(fref);
        }
    }

and what are you going to do with these snapshots? is it a kind of a spy
camera? ;-)

sz.
```

9.6 Inhalt der CD

Auf der beigelegten CD befindet sich:

- Diplomarbeit.pdf — Diplomarbeit als PDF
- Zusammenfassung.pdf — Zusammenfassung auf Deutsch
- Abstrakt.pdf — Zusammenfassung auf Englisch

Ordner:

- Alte Versionen — Backupkopien der Applikation zu verschieden Zeitpunkten
- Bilder und Diagramme — Bilder und Diagramme, welche für diese Arbeit erstellt wurden
- Documentation — verschiedene Dokumentationen und APIs zu Palm und PalmSource
- Orginal — Sourcecode der Vorgänger
- Quellen — Einige verwendete Quellen als PDF
- ROMs — Tools für eigenes ROM zu erstellen und Palm Softwareupdates
- Sourcecode — Quellcode zu der Applikation
  Die Projekte wurden mit Codewarrior for Palm v9.3 kreiert.
- Tools — Tools für den Treo 650