HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Bachelorarbeit

JTourLive
www.tourlive.ch

Stephan Hauser, Michael Wagner

16. Februar - 12. Juni 2009

Experte: Dr. Th. Siegenthaler, CSI Consulting AG
Hochschule für Technik Rapperswil

Aufgabenstellung

Dank Nutzung des seit 2001 in Autos mit Benzinmotor standardmässig eingebauten On-Board Diagnose Bus (OBD-II) kann die cnlab TourLive Symbian Anwendung (TourLive) ohne Zusatzelektronik Fahrdaten wie beispielsweise Motordrehzahl, Gaspedalstellung, Benzinverbrauch erfassen und mit GPS- und Bilddaten kombinieren. Die Online-Darstellung dieser Fahrinformationen über die EcoHelper-Webplattform www.tourlive.ch/ecohelper/ wird bei der Fahrausbildung genutzt, um die Fahrweise im Hinblick auf die Reduktion des Benzinverbrauchs zu optimieren.

Die bestehende TourLive Anwendung ist nur für Symbian-Mobiltelefone von Nokia verfügbar. Im Rahmen dieser Bachelorarbeit ist TourLive zu erweitern und auf Java Plattformen bzw. Mobiltelefone mit Java Unterstützung zu portieren. Die lokale Datenauswertung und Visualisierung soll verbessert und die Erfassung des Fahrparameters "Schubabschaltung" soll optimiert werden. Durch die Realisierung der neuen TourLive Java Anwendung (JTourLive) soll sowohl die Funktionalität von TourLive als auch das EcoHelper-System einem weiteren Nutzerkreis zugänglich gemacht werden.

Acknowledgements

Projektbetreuung

Wir danken Herrn Heinzmann für seine innovativen Ideen und Vorschläge für die Umsetzung unserer Bachelorarbeit. Er stand uns während der gesamten Projektdauer mit grosser Begeisterung für das Thema mit Rat und Tat zur Seite. Er hat sich trotz vielen anderen Verpflichtungen immer wieder viel Zeit für Sitzungen und Problembesprechungen genommen.

Industriepartner

Als Industriepartner für das JTourLive-Projekt stand uns René Vogt von der cnlab AG zur Seite. Mit ihm hatten wir für technische Anliegen eine sehr kompetente Anlaufstelle. Als Entwickler des Symbian TourLive-Systems kannte er die Probleme dieser Domäne aus erster Hand und konnte uns so immer weiter helfen oder vor bösen Überraschungen bewahren. Auch für die Installation der Server-Komponenten auf dem Server der cnlab AG konnte er uns optimal unterstützen.

Verkehrssicherheitszentrum Betzholz

Speziell möchten wir uns auch beim TCS Verkehrssicherheitszentrum (VSZ) Betzholz für die Testmöglichkeit bedanken. Wir durften unsere Anwendung mit verschiedenen Fahrzeugen des VSZ testen.

Externe Meinungen

Wir möchten uns noch bei unseren Kommilitonen Omid Afshari und Thomas Scheuchzer für ihre unabhängigen Meinungen bedanken. Während der Bachelorarbeit konnten wir bei ihnen immer wieder externe, unvoreingenommene Kritik einholen. Dies gab uns immer wieder neue Inputs für die Umsetzung und bewahrte uns vor Projektblindheit.

Korrekturlesen

Für ihre Einsätze beim minutiösen Korrekturlesen der gesamten Arbeit möchten wir uns bei Sarah Schwerzmann und Antoinette Hauser bedanken.

Erklärung[1]

Gegenstand, Leistung	Person	Funktion
Bachelorarbeit	Stephan Hauser	Autor der Arbeit
Bachelorarbeit	Michael Wagner	Autor der Arbeit
Korrekturlesen, Hinweise zur sprachlichen Überarbeitung	Sarah Schwerzmann	Korrekturleserin
Idee, Aufgabenstellung, allgemeines Pflichtenheft, Betreuung während der Arbeit	Prof. Dr. P. Heinzmann	Verantwortlicher Professor
Entwicklung und zu Verfügungstellung TourLive Prototyp (Symbian Version), Betreuung während der Arbeit	R. Vogt, cnlab AG	Industriepartner

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit gemäss obiger Zusammenstellung selber und ohne weitere fremde Hilfe durchgeführt habe,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.

Rapperswil, den 5. Juni 2009 .....
Stephan Hauser

Rapperswil, den 5. Juni 2009 .....
Michael Wagner

[1] Diese Erklärung basiert auf der Mustererklärung in den Richtlinien der HSR zur Durchführung von Projekt-, Studien-, Diplom- oder Bachelorarbeiten vom 16. Februar 2009.

Abstract

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Kurzfassung der Bachelorarbeit

Abteilung	**Informatik**
Name der Studierenden	**Stephan Hauser** **Michael Wagner**
Studienjahr	**FS 09**
Titel der Studienarbeit	**TourLive**
Examinatorin / Examinator	**Peter Heinzmann**

Kurzfassung der Studienarbeit

Die Aufgabe der TourLive Applikation ist es, aus verschiedenen Quellen Daten zu sammeln und diese aufzuzeichnen (Offline-Modus) bzw. an einen Server zu übermitteln (Online-Modus).
Zu den Daten Quellen gehören die OnBoard-Diagnose-Schnittstelle (OBD-II) von PKWs, das Global Positioning System (GPS) und Pulsgurte. Die Verbindung zur OBD-II-Schnittstelle und zum Pulsgurt wird mittels Bleutooth realisiert. GPS Daten können von einem Geräte-internen oder von einem externen GPS-Modul bezogen werden.
Im Online-Modus werden die gesammelten Daten an den TourLive-Server der cnlab AG übermittelt, auf welchem die Daten durch verschiedene Webapplikationen statistisch ausgwertet werden können.
Im Offline-Modus werden die Daten direkt in den Speicher des Aufzeichnungsgerätes geschrieben.

Ein typischer Anwendungsfall von TourLive ist das Sammeln und Übermitteln an den TourLive-Server von OBD-II-Daten eines fahrenden PKWs mit den dazugehörigen GPS-Positionsdaten. Die OBD-II-Daten enthalten Parameter wie Geschwindigkeits-, Drehzahl-, Treibstoffverbrauchs-, und Distanzwerte. Zusammen mit den GPS-Daten kann so auf dem Server die Fahrt inklusive der aktuellen Position auf einer Landkarte genau analysiert werden.

Während des Sammelns der Daten kann TourLive die Werte direkt auf dem Display des Aufzeichnungsgerätes anzeigen oder in Diagrammen grafisch darstellen.

Die TourLive Applikation läuft auf herkömmlichen Computern wie auch auf mobilen Geräten wie Mobiltelefonen oder PDAs. Die TourLive Applikation wurde in Java realisiert, wobei die Desktop Variante mit JavaSE und die mobile Variante mit JavaME entwickelt wurde.

TourLive wurde mit der cnlab AG als Industriepartner entwickelt. Getestet wurde TourLive in Zusammenarbeit mit dem TCS Verkehrssicherheitszentrum Betzholz in Hinwil, welches auch ihre Fahrzeuge zur Verfügung stellten.

Management Summary

Ausgangslage

Die Aufgabe der JTourLive Anwendung ist es, mit Mobiltelefonen oder Notebooks aus verschiedenen Quellen Daten zu sammeln und diese aufzuzeichnen (Offline-Modus) bzw. an einen Server zu übermitteln (Online-Modus). Zu den Datenquellen gehören die OnBoard-Diagnose-Schnittstelle (OBD-II) von PKWs, das Global Positioning System (GPS) und ein Pulsgurt. Die Verbindung zur OBD-II-Schnittstelle und zum Pulsgurt wird mittels Bluetooth realisiert. GPS Daten können von internen oder externen GPS-Modulen bezogen werden. Im Online-Modus werden die gesammelten Daten an den TourLive-Server der cnlab AG übermittelt, auf welchem die Daten durch verschiedene Webanwendungen statistisch ausgewertet und visualisiert werden können. Im Offline-Modus werden die Daten direkt in den Speicher des Aufzeichnungsgerätes geschrieben.

Die cnlab AG hat bereits eine Software für Symbian Plattformen (TourLive) entwickelt, welche diese Funktionalität abdeckt. Im Rahmen dieser Bachelorarbeit soll diese Software mit erweitertem Funktionsumfang auf Java portiert werden, um sie einem weiteren Nutzerkreis zugänglich zu machen.

Im Rahmen der vorliegenden Bachelorarbeit ist eine TourLive Java Anwendung (JTourLive) zu realisieren, welche auf möglichst vielen Mobiltelefonen, Personal Digital Assistents (PDA) sowie auch auf Desktop PCs lauffähig ist. Die Anwendung soll die Daten zur EcoHelper-Plattform http://www.cnlab.ch/ecohelper/ senden können, so dass sie dort visualisiert und ausgewertet werden können.

Vorgehen

Der Kern der JTourLive Anwendung wurde so entwickelt, dass er sowohl unter JavaSE als auch unter JavaME kompiliert werden kann. Die GUI abhängigen Komponenten wurden jeweils separat in JavaSE für die Desktop Variante und in JavaME für die mobilen Geräte entwickelt.

Für JTourLive wurde die bestehende Symbian TourLive Anwendung als Vorlage verwendet, wobei sich der Funktionsumfang etwas verlagert hat. Einerseits konnten einige Funktionen in Java nicht abgebildet werden, da der direkte Zugriff auf Geräteressourcen über Java APIs eingeschränkt ist. Andererseits wurden neue oder erweiterte Funktionen realisiert, welche in der Symbian Variante nicht vorhanden sind. Dazu gehören z.B. die Korrektur der GPS-Höhe vom WGS84-Format in die effektive Höhe über Meer, eine bessere Erkennung der Schubabschaltung oder der verbesserte Gangerkennungs-Algorithmus.

Die Software wurde in Zusammenarbeit mit der cnlab AG als Industriepartner und mit Prof. Dr. P. Heinzmann als Betreuer realisiert. Getestet wurde die JTourLive Anwendung mit Fahrzeugen des Verkehrssicherheitszentrums (VSZ) Betzholz.

Ergebnisse

Entstanden ist während dieser Bachelorarbeit eine Anwendung mit verschiedenen Modulen, welche wahlweise zu- oder ausgeschaltet werden können. Die Anwendung besitzt ein OBD-,

ein GPS-, ein Puls-, ein Foto-, ein Online Modus-, ein Offline Modus- und ein System-Modul. Die Module verbinden sich automatisch zu ihren externen Peripheriegeräten um die Daten aufzuzeichnen. Bei Unterbrüchen wird die Verbindung automatisch neu aufgebaut.

Die neue JTourLive Anwendung in Java kann also die Symbian Variante ersetzen und die Daten aus den externen Quellen an den TourLive- bzw. EcoHelper-Server der cnlab AG übermitteln. Sie ist in die bestehenden TourLive Systeme (DB, EcoHelper Webseite, TourLive Webseite) integriert. Lediglich für die serverseitige Entgegennahme der Daten wurde eine neue Server-Komponente in PHP entwickelt.

Die vorliegende Version von JTourLive wurde vorerst nicht signiert. Dies hat zur Folge, dass auf den Endgeräten vor dem Zugriff auf System-Ressourcen Sicherheitsfragen erscheinen. Die Funktionen sind jedoch vollumfänglich nutzbar.

Ausblick

Die neue JTourLive Anwendung ist voll funktionsfähig und soll nun bei den Verkehrszentren getestet werden. Bei ersten Tests im TCS Verkehrssicherheitszentrum Betzholz in Hinwil äusserten die Fahrlehrer den Wunsch, während der Fahrt noch Videoaufnahmen vom Fahrer und von der Fahrbahn aufzeichnen zu können. Mit den Erfahrungen aus den ausgiebigen Tests und nach allfälligen GUI-Verbesserungen wird die cnlab AG JTourLive anpassen und einem weiteren Nutzerkreis zugänglich machen.

Inhaltsverzeichnis

1 Einführung und Übersicht

1.1 Was ist TourLive?

TourLive ist ein System der cnlab AG, welches Daten aus unterschiedlichen Quellen sammeln und an eine zentrale Datensenke übermitteln kann. Als Datenquellen werden zur Zeit die OnBoard-Diagnose-Schnittstelle (OBD-II) von Personenwagen, GPS-Module, Pulsgurte, Mobilfunkzellen und in die mobilen Geräte integrierte Kameras eingesetzt. Als Datensenke dient ein Server der cnlab AG, welcher die übertragenen Daten in einer Datenbank ablegt und verschiedenen Webanwendungen zur Analyse und grafischen Aufbereitung zur Verfügung stellt.

1.2 Ausgangslage

Das TourLive-System wurde von der cnlab AG entwickelt um den Verlauf von Radrennen und Läufen "live" ins Internet zu übertragen. Die Aufzeichnungsgeräte werden dabei von Begleitfahrzeugen oder direkt von den Sportlern mitgeführt. Mit den so übermittelten GPS-Positionen und Geschwindigkeiten kann z.B. der Rückstand vom Feld auf die Spitzengruppe berechnet oder der Verlauf des Rennens direkt auf einer Karte dargestellt werden. In weiteren Schritten wurde das TourLive-System ausgebaut um weitere Werte aus der OnBoard-Diagnose-Schnittstelle (OBD-II) von Personenwagen und Pulsdaten von Pulsgurten zu sammeln. So wurde das Einsatzgebiet des TourLive-Systems auf das Analysieren von Autofahrten erweitert.

Die Webanwendung www.tourlive.ch bietet die Möglichkeit die aufgezeichneten Fahrten einzusehen, die gesammelten Werte in einfachen Diagrammen darzustellen und die gefahrene Strecke auf einer Karte anzuzeigen.

EcoHelper (www.tourlive.ch/ecohelper/), eine weitere Webanwendung, welche als Vorgänger-Studienarbeit zu dieser Bachelorarbeit verfasst wurde, kann aufgezeichnete Fahrten detailliert analysieren, vergleichen und in kombinierten Diagrammen darstellen. EcoHelper wird hauptsächlich eingesetzt um Potential in der Fahrstiloptimierung hinsichtlich des Kraftstoffverbrauchs zu ermitteln.

Die von der cnlab AG bis dahin eingesetzte TourLive Anwendung wurde für Aufzeichnungsgeräte mit Symbian Plattform entwickelt und nur mit bestimmten Nokia Mobiltelefonen betrieben.

1.3 Umfang dieser Arbeit

Im Rahmen dieser Bachelorarbeit soll ein neues TourLive Aufnahmesystem entwickelt und in die unter Abschnitt 1.2 auf der vorherigen Seite beschriebene Systemlandschaft integriert werden. Die bis dahin für Symbian Plattformen entwickelte TourLive Anwendung soll unter dem Namen JTourLive auf Java portiert und im Funktionsumfang erweitert werden. Das oberste Ziel ist die Plattformunabhängigkeit, um das Aufnahmesystem einem grösseren Nutzerkreis zur Verfügung zu stellen. Weiter soll die Anzeige der gesammelten Werte direkt auf dem Aufnahmegerät erweitert werden.

Als Datenquellen müssen die OBD-II-Schnittstelle, GPS-Module, Pulsgurte und geräteinterne Kameras unterstützt werden. Bei den OBD-Daten soll die Erkennung der Schubabschaltung und die Erkennung der Gänge verbessert und bei den GPS-Daten die Höhe vom WGS84-Format auf die exakte Höhe über Meer korrigiert werden.

Da im Vergleich zur bisherigen TourLive Anwendung neue zusätzliche Daten an den Server übermittelt werden, müssen auch die serverseitigen Komponenten erweitert werden. Neu soll für die Server-Seite eine Anwendung entwickelt werden, welche ein automatisches Update der JTourLive Anwendung vom mobilen Gerät aus direkt über das Internet ermöglicht.

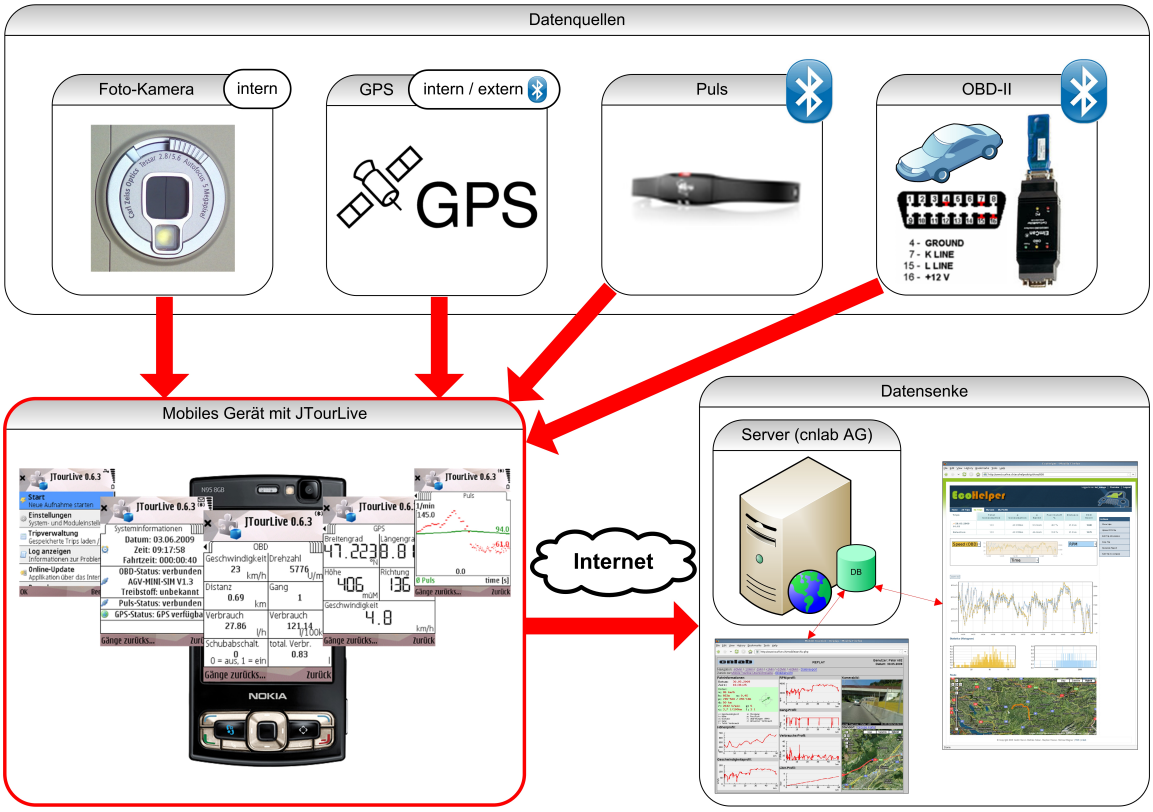

Abbildung 1.1: Übersicht: JTourLive Anwendung

1.4 Wie ist diese Dokumentation aufgebaut?

In Kapitel 2 auf der nächsten Seite werden die Ergebnisse aus der Analyse-Phase beschrieben. Es ist das ausführlichste Kapitel, da das Erstellen verschiedener Prototypen und Machbarkeitsstudien ein Grossteil dieser Bachelorarbeit ausmachen.

In Kapitel 3 auf Seite 25 wird die schlussendliche Realisierung der JTourLive Anwendung beschrieben. Dieses Kapitel wendet sich vorwiegend an Entwickler, welche am JTourLive Projekt interessiert sind oder es weiter entwickeln wollen.

In Kapitel 4 auf Seite 38 wird der Einsatz und mögliche Weiterentwicklungen beschrieben. Dieses Kapitel richtet sich an Interessenten von JTourLive.

Unter Kapitel A auf Seite 41 befindet sich die Benutzeranleitung zu JTourLive. Sie richtet sich in erster Linie an Anwender von JTourLive.

Entwickler finden unter Kapitel C auf Seite 56 einen Anleitung zur Einrichtung der Entwicklungsumgebung, um JTourLive weiterentwickeln zu können.

2 Analyse

Für die Portierung von TourLive auf Java musste vorgängig für jede Funktionalität die Realisierbarkeit abgeklärt werden. Da im Bereich von Java und vor allem JavaME kein Wissen vorhanden war, wurden alle APIs für den Zugriff auf die unterschiedlichen Ressourcen zu Beginn dieser Arbeit analysiert und dokumentiert. Mit Prototypen wurde abgeklärt, welche Module für JTourLive umgesetzt werden können und welche nicht. Die Ergebnisse dieser Analysephase sind in diesem Kapitel zusammengefasst.

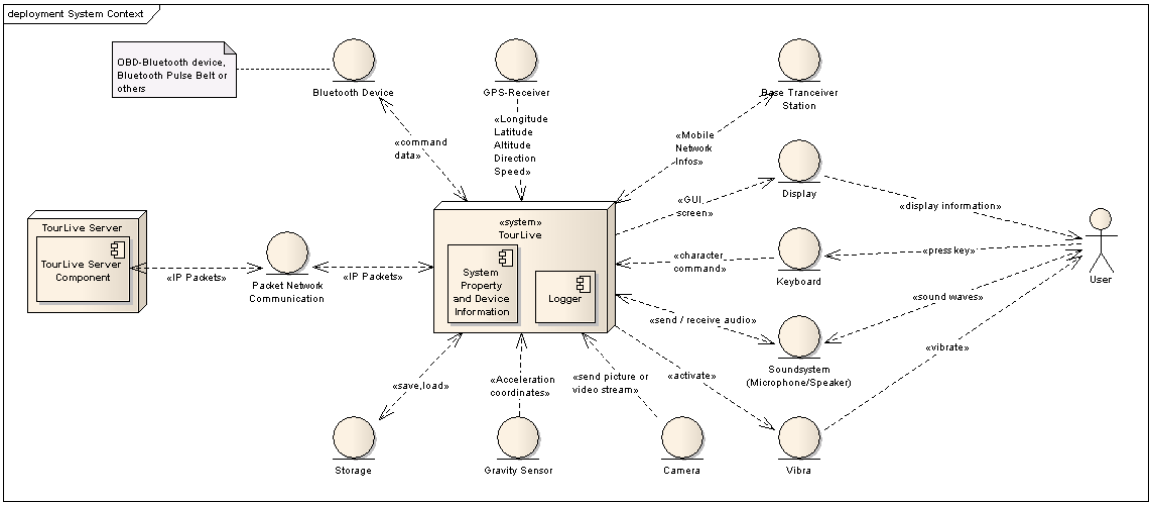

Abbildung 2.1: Ressourcen im System Context von JTourLive

2.1 Location API

Das Location API wird für JavaME im JSR179[18] spezifiziert. Es abstrahiert von der zugrunde liegenden Lokalisierungstechnik und unterstützt die Entwicklung von sogenannten Location Based Services (LBS).

Lokalisierungsmethoden

Die in der Spezifikation abgedeckten Lokalisierungsmethoden sind folgende:

MTE_SATELLITE Die klassische Lokalisierung mittels GPS oder Galileo Satelliten

MTE_TIMEDIFFERENCE Berechnung der Position anhand der Differenz verschiedener GSM- oder UMTS-Signallaufzeiten

MTE_TIMEOFARRIVAL Berechnung der Position mittels GSM- oder UMTS-Signallaufzeitmessung

MTE_CELLID Position wird anhand des Standortes der aktuellen Base Transceiver Station (BTS) ermittelt.

MTE_SHORTRANGE Ermittlung der Position anhand von Kurzstreckenfunknetzen wie zum Beispiel Bluetooth oder WLAN.

MTE_ANGLEOFARRIVAL Die Position wird durch eine Sektorzelle[1] und der GSM- oder UMTS-Signallaufzeit ermittelt.

Positionsinformationen

Die *Location* Klasse kapselt die Informationen einer Position. Folgende Positionsdaten werden unterstützt, stehen jedoch abhängig von der Lokalisierungsmethode nicht immer alle zur Verfügung:

Adressinformationen Informationen über den aktuellen Ort des mobilen Gerätes als *String*, wenn diese vorhanden sind, sonst *null*.

Orientierung Die Himmelsrichtung als *Float* im Intervall [0..360] in Bezug auf Norden, sonst *Float.NaN*.

Zusatzinformationen Weitere Informationen inklusive MIME-Typ.

Lokalisierungsmethode Welche der oben erwähnten Lokalisierungsmethoden für die Gewinnung der Positionsinformationen eingesetzt wurden.

Koordinaten Längengrad, Breitengrad, Höhe über Normal Null[2] im WGS84-Format[13] und deren Genauigkeit.

Geschwindigkeit Die Geschwindigkeit, die das mobile Gerät seit der letzten gemessenen Position im Schnitt hatte in Metern pro Sekunde.

Zeitstempel Der Zeitpunkt der Positionsbestimmung in Millisekunden.

Positionskoordinaten

Die Position des mobilen Gerätes wird in der *Coordinates* Klasse gekapselt. Über sie können die effektiven Positionsdaten im WGS84-Format[13] gewonnen werden.

Höhe Die Höhe über Normal Null in Metern im WGS84-Format. Diese Höhe wird in JTourLive auf die effektive Höhe über Meer, wie sie auf Landkarten angegeben wird, korrigiert. Mehr dazu im nächsten Abschnitt 2.1.1 auf der nächsten Seite Genauigkeit der GPS-Höhenangabe.

Breitengrad Der Breitengrad im Intervall [-90.0,90.0] in Grad, Minuten und Sekunden.

Längengrad Der Längengrad im Intervall [-180.0,180.0] in Grad, Minuten und Sekunden.

[1] Eine Sektorzellen-Antenne sendet ihr Signal nicht kreisförmig aus sondern leuchtet nur einen bestimmten Sektor des Kreises aus.

[2] Die Höhenbezugsfläche repräsentiert die mittlere Meereshöhe als ein Referenzellipsoid das die Erdoberfläche annähert.

2.1.1 Genauigkeit der GPS-Höhenangabe

Die GPS-Daten werden vom GPS-Empfänger in mobilen Geräten im WGS84-Format[13] geliefert. Die GPS-Höhe wird als Höhe in Metern über Normal Null angegeben. Normal Null entspricht dabei der Referenzfläche des Ellipsoides, welches die Form der Erde annähert. Dieses Ellipsoid wird bei Höhenangaben über Normal Null als Näherung des Meeresspiegels beigezogen. Die Erde entspricht aber in Wirklichkeit nicht genau diesem Ellipsoid. Durch verschiedene Dichten des Erdmaterials und die daraus resultierenden unterschiedlich starken Gravitationskräfte ist die Erde eigentlich ein verbeultes Ellipsoid; eine Art "Kartoffel". Diese "Kartoffel" wird Geoid[3] genannt.

Um jetzt auf die exakte Höhe, wie sie etwa auf Landkarten zu finden ist, zu kommen, muss die GPS-Höhe um einen bestimmten Wert korrigiert werden. Dieser Korrekturwert entspricht der Höhendifferenz der beiden Körper (Ellipsoid und Geoid) an einer bestimmten Position. Da das Geoid keine Regelmässigkeit aufweist muss diese Höhendifferenz für jede Position wieder neu berechnet werden. Der Korrekturwert schwankt dabei zwischen -106 bis + 85 Metern[21].

Das Geoid wird durch das EGM96 Modell dargestellt, welches in Zusammenarbeit der National Imagery and Mapping Agency (NIMA)[23], des NASA Goddard Space Flight Center (GSFC)[22] und der Ohio State University entwickelt wurde. Das EGM96 Geoid-Modell wird durch eine Formel mit mehreren tausend Koeffizienten dargestellt und nähert so an die Form des Geoides an. Die Berechnung des EGM96 Modells ist für eine JavaME Anwendung zu komplex und viel zu zeitintensiv, wenn im Sekundentakt GPS-Daten geliefert werden sollen. Aus diesem Grund wird für die JTourLive Anwendung eine Tabelle mit vor-kalkulierten Korrekturwerten verwendet. Die Werte wurden mit dem Geoid Height Calculator http://sps.unavco.org/geoid/ von UNAVCO[10] für jedes Längen- und Breitengrad berechnet.

Auflösung	Anzahl Werte	JavaME .class Dateigrösse
1-Grad-Raster	65341	276.8 KB
2-Grad-Raster	16471	74.5 KB
3-Grad-Raster	7381	44.2 KB
4-Grad-Raster	4186	25.9 KB
5-Grad-Raster	2701	17.4 KB
6-Grad-Raster	1891	12.8 KB

Tabelle 2.1: Dateigrössen bei den verschiedenen Auflösungen

Da eine JavaME Klasse im kompilierten Zustand 32KB nicht überschreiten darf, musste die Auflösung der Korrekturdaten reduziert werden. Eine Reduktion auf ein 4-Grad-Raster würde von der Dateigrösse her ausreichen, für die Durchschnittsbildung und die anschliessende Interpolation ist es jedoch einfacher, wenn auf ein ungerades Grad-Raster reduziert wird. Eine ungerade Zahl hat den Vorteil, dass es in der Mitte der zusammengefassten Werte genau eine Zahl hat. Aus diesem Grund wurden die Korrekturdaten für die JTourLive Anwendung auf ein 5-Grad-Raster reduziert. Die Grösse der Java Klasse wurde dadurch um den Faktor 25 kleiner. Der durchschnittliche quadratische Fehler vom 5-Grad-Raster

[3] Geoid könnte zu deutsch auch als Erdform übersetzt werden.

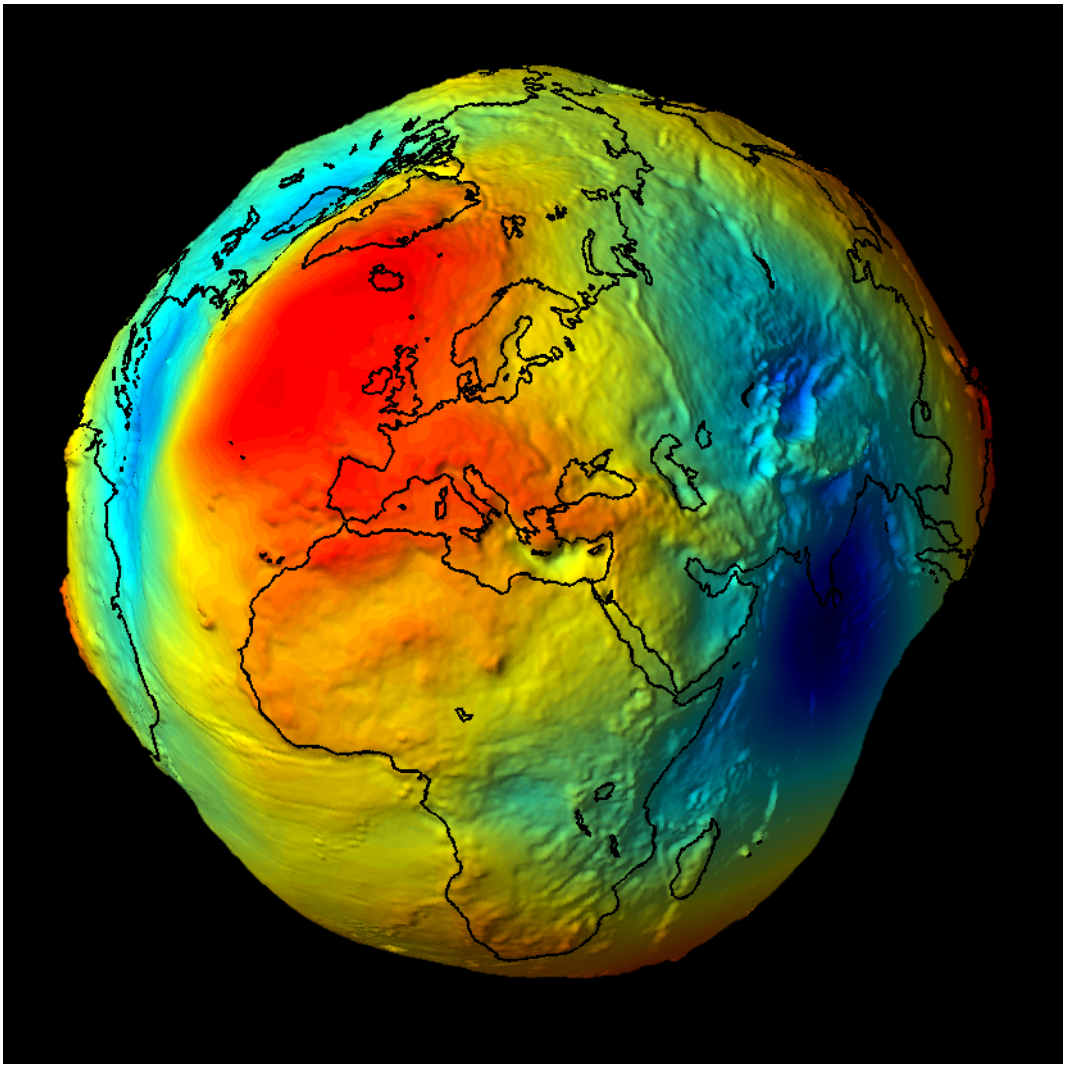

Abbildung 2.2: Geoid [9]

beträgt 31.3, wenn nur jeder fünfte Wert genommen wird, ohne die weggelassenen Werte zu berücksichtigen. Mit dieser Methode entstand eine durchschnittliche Abweichung von über 5 Metern.

Auflösung	ohne Durchschnitt	mit Durchschnitt	mit Interpolation
1-Grad-Raster	2.1	2.1	2.1
2-Grad-Raster	4.1	2.1	2.4
3-Grad-Raster	11.1	4.4	2.1
4-Grad-Raster	20.2	7.2	4.8
5-Grad-Raster	31.3	10.3	5.4
6-Grad-Raster	45.4	14.0	9.0

Tabelle 2.2: Durchschnittliche quadratische Fehler in Meter

Der durchschnittliche quadratische Fehler konnte mittels Durchschnittsbildung über 5 Werte bei der Auflösungsreduktion auf 10.3 reduziert werden. Dies entspricht nur noch einer Abweichung von durchschnittlich gut 3.2 Metern.

Der durchschnittliche quadratische Fehler konnte durch gewichtetes Auslesen (Interpolation) über bis zu vier Werte nochmals um knapp die Hälfte auf 5.4 reduziert werden. Die Abweichung beträgt dadurch durchschnittlich nur noch 2.3 Meter. Dies erfüllt die Anforderung von einem maximalen durchschnittlichen Fehler von 5 Metern immer noch gut.

2.2 OBD-II / EOBD Schnittstelle

OBD-II wurde im Jahr 1996 in Amerika standardisiert und ist seit diesem Zeitpunkt für alle Autos Pflicht. 1998 wurde in Europa die EOBD (European OnBoard Diagnosis) Spezifikation veröffentlicht, welche nur kleine Abweichungen zu OBD-II hat. Seit 2001 müssen auch in Europa alle PKWs mit Benzinmotoren für die Zulassung eine solche Schnittstelle anbieten. PKWs mit Dieselmotoren erst seit 2004.

Über die OBD-II-Schnittstelle können Diagnosedaten aller Abgas-beeinflussenden Systeme ausgelesen und teilweise auch gesteuert werden. Die physikalische Verbindung die dafür verwendet wird, ist unter SAE J1850 bzw. ISO 9141-2 normiert. Der entsprechende Stecker muss vom Fahrersitz, maximal 50cm vom Steuerrad entfernt, erreichbar sein. Über diese Schnittstelle kommuniziert der Controller über eines der unter ISO 9141-2, ISO 14230-4, ISO 15765-4, SAE J1850-PWM und SAE J1850-VPW definierten Protokolle mit dem Diagnosegerät.

Um die Komplexität, die durch diese unterschiedlichen Protokolle entsteht, zu verstecken, wird das ElmCanII Interface ELM327[5] verwendet. Dieses Interface abstrahiert alle Spezialfälle, vereinheitlicht die Initialisierung, ermöglicht einen einfachen Text-basierten Zugriff auf die Daten und vereinfacht die Programmierung durch Fehlerbehandlung und weitere geeignete Hilfsmittel.

An die Serielle Schnittstelle des ElmCanII Interfaces wird ein Bluetooth Dongle angeschlossen, welches per RFCOMM die Daten über Bluetooth an das mobile Gerät mit der JTour-Live Anwendung weiterleitet.

2.2.1 PIDs

Der OBD-II-Standard definiert sogenannte PIDs (Parameter IDs, auch P-Codes genannt) [35]. Diese IDs definieren die Nachrichten, die von einem OBD-II-Controller verarbeitet und beantwortet werden können.

Jede Nachricht besteht dabei aus zwei Teilen. Im ersten Byte ist der sogenannte Mode zu finden, der dem Controller sagt, was er mit der Nachricht zu tun hat. Momentan sind 10 Modi definiert, wovon zwei für uns interessant sind:

Mode $01 Show Current Data (Zeigt Abgassystem-relevante Daten)

Mode $09 Request Vehicle Information (Zeigt allgemeine Daten zum Auto an)

Das zweite Byte der Nachricht ist dann die PID, welche je nach Mode eine unterschiedliche Bedeutung hat. Eine Auflistung aller standardisierten PIDs ist unter [35] zu finden.

2.2.2 ELM327 Monitoring Mode

Der ELM327 Controller kann mit Hilfe des Befehls *AT MA* in einen Bus Monitoring Mode versetzt werden, in dem er jegliche Kommunikation auf dem OBD-II-Bus weiterleitet. In diesem Mode können keine Befehle abgesetzt werden und der Controller sendet keine Bestätigungen und Keep-Alive Nachrichten mehr ab.

Der Monitoring Mode kann sehr nützlich sein, da über den Bus oft Nachrichten versendet werden, die nicht über die normalen PIDs abgefragt werden können (z.B. Bremsen). Das rührt daher, dass OBD-II nur für Abgas-relevante Systeme entwickelt wurde. Da während dem Monitoring Mode keine Nachrichten gesendet werden können, können in diesem Modus keine eigenen Anfragen gesendet werden.

2.3 Gangerkennung

Im OBD-II Standard SAE J/1979 ist keine Parameter-ID (PID) für die aktuelle Gangposition vorhanden. Aus diesem Grund muss die Gangposition aus dem Verhältnis von Geschwindigkeit und Motordrehzahl erkannt werden. Damit alle Gänge sauber erkannt werden können, muss auch jeder Gang durchgeschaltet und gefahren werden. Es ist unmöglich den zweiten Gang als zweiten Gang zu erkennen, wenn direkt in diesem angefahren wird. Auch gibt es keine Möglichkeit den Rückwärtsgang als Rückwärtsgang zu erkennen, da OBD-II die Geschwindigkeit auch beim rückwärts fahren positiv angibt. Wenn der Rückwärtsgang gleich übersetzt ist wie z.B. der erste Gang, wird dieser auch als erster Gang erkannt. Wenn der Rückwärtsgang eine andere Übersetzung hat, wird er als zusätzlichen Gang erkannt.

2.3.1 Gangerkennungs-Algorithmus auf dem mobilen Gerät

Das mobile Gerät muss über einen Gangerkennungs-Algorithmus verfügen, welcher von Beginn weg die Gänge fortlaufend erkennt. Das bedeutet aber, dass wenn beim ersten Anfahren ein Gang übersprungen wird, die darauf folgenden Gänge um ein Gang versetzt erkannt werden, bis der übersprungene Gang einmal gefahren wird.

Während ein Gang gefahren wird, bleibt das Verhältnis von Geschwindigkeit und Motordrehzahl immer gleich. Da über die OBD-II-Schnittstelle immer nur ein Wert gleichzeitig abgefragt werden kann, müssen Geschwindigkeit und Motordrehzahl mit einer Verzögerung von bis zu 200ms sequentiell abgefragt werden. So erhält man Wertepaare, die zeitlich nicht exakt zueinander gehören. Dies hat zur Folge, dass je nach Beschleunigung des Fahrzeuges das errechnete Verhältnis von Geschwindigkeit und Motordrehzahl schwanken kann, obwohl immer im selben Gang gefahren wird. Der Gangerkennungs-Algorithmus muss also eine gewisse Toleranz haben, mit welcher er das Verhältnis einem Gang zuordnet.

Weiter muss das Verhältnis von Geschwindigkeit und Motordrehzahl über eine gewisse Zeit konstant bleiben um als Gang erkannt werden zu können, da man sonst während dem Kuppeln weitere nicht vorhandene Gänge erkannt hätte. Die Erkennungsquote war am höchsten bei einem konstanten Verhältnis während 12 Abfrage-Iterationen. Das entspricht ca. 6s.

Dazu kommt noch, dass bei einer Geschwindigkeit von 0km/h kein Gang erkannt werden darf, da man sonst das Standgas als Gang erkennen würde.

Mit den oben erwähnten Massnahmen können die Gänge sehr zuverlässig erkannt werden. Das einzige Problem welches bleibt, ist die Erkennung des Rückwärtsgangs. Wenn er weniger als die oben erwähnten 6s lang gefahren wird, wird er nicht erkannt und stellt kein Problem dar. Wenn er aber länger gefahren wird, wird er je nach Übersetzung als erster oder zweiter Gang erkannt. Dieses Problem kann bei der kontinuierlichen Erkennung auf dem mobilen Gerät nicht behoben werden. Der Rückwärtsgang kann höchstens serverseitige mit einer Analyse des gesamten Datensatzes einer Fahrt erkannt werden.

2.3.2 Serverseitige Gangerkennung

Im Gegensatz zum Algorithmus auf dem mobilen Gerät, welcher vom Beginn der Fahrt an die Gänge fortlaufend erkennen muss, hat man auf der Seite des Servers die Möglichkeit eine abgeschlossene Fahrt als ganzen Datensatz zu analysieren. Dort könnte auch mit entsprechender Logik der Rückwärtsgang erkannt werden, da dieser meistens nur am Anfang oder am Ende einer Fahrt gebraucht wird. Auch die restlichen Gänge können über die gesamte Fahrt hinweg analysiert werden. So ist man serverseitig nicht darauf angewiesen, dass beim ersten Anfahren jeder Gang sauber durchgeschaltet wird.

2.4 Schubabschaltung

Wenn ein Fahrzeug rollt, ein Gang eingelegt ist, jedoch kein Benzin eingespritzt wird, nennt man das Schubabschaltung. Moderne Fahrzeuge haben dann einen Verbrauch von 0 Litern. Die Erkennung der Schubabschaltung kann anhand des *Fuel System Status (PID 03)* erkannt werden. Das dritte Bit des zurückgelieferten Wertes ist gesetzt, wenn die Schubabschaltung aktiv ist oder wenn der Motor unter Last ist. Es gilt also diese zwei Situationen noch zu unterscheiden, da sonst fälschlicherweise auch das Beschleunigen eines Fahrzeuges als Schubabschaltung erfasst würde. Zu diesem Zweck wurde ein Algorithmus entwickelt, welcher zusätzlich noch die Werte der Motorenlast und der Gaspedalposition auswertet. Wenn also das dritte Bit des *Fuel System Status* gesetzt ist und die Gaspedalposition oder die Motorenlast tief ist, wird eine aktive Schubabschaltung erkannt.

2.5 Puls und Kalorien

Der Kalorienverbrauch lässt sich nicht einfach so berechnen. Er hängt von vielen Faktoren ab. Unter diesen Faktoren gibt es fix bestimmbare Werte wie Puls, Alter, Gewicht, Körpergrösse und Geschlecht, aber auch nicht genau bestimmbare Werte, wie Belastungsintensität, Fitness-Niveau oder genetische Veranlagungen. Letztere können nicht in die Berechnung einfliessen, da sie nicht in Zahlen ausgedrückt werden können. Deshalb ist die Kalorien-Verbrauchsberechnung immer eine Näherung und keinen falls eine exakte Berechnung. Trotzdem gibt sie der Sport-treibenden Person die Möglichkeit ihren Kalorienverbrauch abzuschätzen.

2.5.1 Basal Metabolic Rate (BMR)

Die Basal Metabolic Rate (BMR) bezeichnet den Grundumsatz eines menschlichen Körpers während 24 Stunden. Die BMR wird seit 1919 mit der Harris-Benedict Gleichung[33] berechnet. Die Harris-Benedict Gleichung hat für Männer und Frauen unterschiedliche Koeffizienten. Weiter fliessen Körper-Koordinaten wie das Gewicht in Kilogramm, die Höhe in Zentimeter und das Alter in Jahren in die Berechnung mit ein.

Algorithm 2.1 Harris-Benedict Gleichung

$BMR_{Mann}[kcal\,Tag^{-1}] = 66.4730 + 13.7516[mkg^{-1}] * m_{Körper}[kg] + 5.0033[cm^{-1}] * h_{Körper}[cm] - 6.7550[Jahr^{-1}] * t_{Alter}[Jahr]$

$BMR_{Frau}[kcal\,Tag^{-1}] = 655.0955 + 9.5634[mkg^{-1}] * m_{Körper}[kg] + 1.8496[cm^{-1}] * h_{Körper}[cm] - 4.6756[Jahr^{-1}] * t_{Alter}[Jahr]$

1990 haben Mifflin und St. Jeor eine neue Formel entwickelt, bei welcher nur ein Koeffizient für das Geschlecht geändert werden muss. Die Formel entspricht auch eher dem heutigen Lebensstandard und liefert 5% genauere Werte als die Harris-Benedict Gleichung verglichen mit exakten Messungen.

Algorithm 2.2 Mifflin - St Jeor Formel

$BMR_{Mann}[kcal\,Tag^{-1}] = 9.99[mkg^{-1}] * m_{Körper}[kg] + 6.25[cm^{-1}] * h_{Körper}[cm] - 4.92[Jahr^{-1}] * t_{Alter}[Jahr] + 5$

$BMR_{Frau}[kcal\,Tag^{-1}] = 9.99[mkg^{-1}] * m_{Körper}[kg] + 6.25[cm^{-1}] * h_{Körper}[cm] - 4.92[Jahr^{-1}] * t_{Alter}[Jahr] - 161$

2.5.2 Physical Activity Level (PAL)

Als Physical Activity Level (PAL) wird die Körperaktivität bezeichnet. In der Tabelle 2.3 auf der nächsten Seite sind PAL-Werte aus dem Alltag aufgeführt. Der PAL-Wert wird mit der BMR multipliziert, um den Gesamt-Kalorienverbrauch an einem Tag zu berechnen.

Algorithm 2.3 Tageskalorienumsatz

$E_{normal}[kcal] = PAL * BMR$

Körperaktivität	Beispiele	PAL_{normal}
Ausschliesslich sitzende oder liegende Lebensweise	alte, gebrechliche Menschen	1.2
Ausschliesslich sitzende Tätigkeit mit wenig oder keiner anstrengenden Freizeitaktivität	Büroangestellte, Feinmechaniker	1.4 - 1.5
Sitzende Tätigkeit, zeitweilig auch zusätzlicher Energieaufwand für gehende und stehende Tätigkeiten	Laboranten, Kraftfahrer, Studierende, Fliessbandarbeiter	1.6 - 1.7
Überwiegend gehende und stehende Arbeit	Hausfrauen, Verkäufer, Kellner, Handwerker	1.8 - 1.9
Körperlich anstrengende berufliche Arbeit	Bauarbeiter, Landwirte, Leistungssportler	2.0 - 2.4

Tabelle 2.3: PAL Tabelle[8]

2.5.3 Kalorienverbrauch bei sportlicher Aktivität

Um die Kalorienmenge, welche während einer sportlichen Aktivität umgesetzt wird, zu berechnen, muss der PAL während der sportlichen Aktivität bekannt sein. In der JTourLive Anwendung wird dieser temporäre PAL während der sportlichen Aktivität aus der Pulsfrequenz abgeleitet, in dem der aktuelle Puls durch 50 dividiert wird.

Algorithm 2.4 temporärer PAL während sportlicher Aktivität

$$PAL_{Sport} = f_{aktuellerPuls}/50$$

Dieser temporäre PAL kann sich während einer sportlichen Aktivität sehr schnell verändern, da er direkt von der aktuellen Pulsfrequenz abhängig ist. Aus diesem Grund darf dieser temporäre PAL nur mit der auf die Zeitspanne gleicher Pulsfrequenz $\triangle t$ herunter gerechneten BMR multipliziert werden.

Algorithm 2.5 temporärer zusätzlicher Kalorienumsatz

$$\triangle E[kcal] = \triangle t[s] * PAL_{Sport} * MBR/(24 * 60 * 60)$$

Dieser temporäre zusätzliche Kalorienumsatz $\triangle E$ kann während einer sportlichen Aktivität aufsummiert werden und so der trainierenden Person den ungefähren zusätzlichen Kalorienumsatz E_{Sport} gegenüber dem Alltagskalorienumsatz anzeigen.

Algorithm 2.6 zusätzlicher Kalorienumsatz durch sportliche Aktivität

$$E_{Sport}[kcal] = \sum \triangle E$$

2.6 Prototypen

2.6.1 Bluetooth OBD

Über die OBD-II-Schnittstelle liefert das Auto Diagnose-Daten wie z.B. die aktuelle Geschwindigkeit oder die Motordrehzahl. Diese Daten werden wie unter Abschnitt 2.2 auf Seite 8 beschrieben über das RFCOMM-Protokoll an das mobile Gerät versendet. Das mobile Gerät kann dadurch über die Bluetooth-Schnittstelle Daten von der OBD-II-Schnittstelle empfangen und Befehle an diese senden. Da das Abfragen dieser Daten je nach Steuergerät des Fahrzeuges einige Zehntelsekunden in Anspruch nimmt, empfiehlt es sich, die Abfragen in einem eigenen Thread durchzuführen, der nur dafür zuständig ist.

Der OBD-Prototyp demonstriert, dass es über das Bluetooth API *javax.bluetooth* möglich ist diese Daten abzufragen, in die gewünschte Form zu bringen und auszuwerten. Die Voraussetzungen sind ein mobiles Gerät mit Bluetooth sowie Unterstützung des MIDP 2.0 API mit dem Paket *javax.bluetooth* des [30]JSR 82.

2.6.2 Bluetooth Puls

Der untersuchte und durch den Prototypen unterstützte Pulsgurt wird von der Firma mobimotion GmbH unter dem Namen Spurty-Brustgurt [20] entwickelt und vertrieben. Der Pulsgurt kommuniziert über das RFCOMM Protokoll über Bluetooth mit einem proprietären Protokoll, welches von der cnlab AG [6] per Reverse Engineering entschlüsselt wurde.

Der JavaME Puls-Prototyp der im Rahmen dieser Bachelorarbeit erstellt wurde, demonstriert, wie über das Bluetooth API *javax.bluetooth* die Zeit zwischen zwei Herzschlägen ausgelesen und dadurch der Puls berechnet werden kann. Auf Basis dieser Daten sowie einigen Angaben zum Benutzer kann dann z.B. auch ein ungefährer Kalorienverbrauch berechnet werden. Mehr zum Thema Puls und Kalorienverbrauchsberechnung unter Abschnitt 2.5 auf Seite 11. Die Voraussetzungen sind ein mobiles Gerät mit Bluetooth sowie Unterstützung des MIDP 2.0 API mit dem Paket *javax.bluetooth* des JSR 82.

Mittlerweile sind weitere Pulsgurte mit Bluetooth-Schnittstelle verfügbar. Nokia bietet in Zusammenarbeit mit Polar ein Bluetooth-Pulsgurt für sein Sports Tracker [25] Programm an. Sony Ericsson bietet einen Bluetooth-Pulsgurt mit integriertem GPS Empfänger [28] an. Und Athlosoft [3] bietet 2 Varianten eines Bluetooth Pulsgurtes für verschiedene Sportarten an. Die Athlosoft Software gibt es ab Sommer 2009 für Windows Mobile, Symbian und iPhone Geräte.

2.6.3 Beschleunigungssensor

Um den Beschleunigungssensor anzusteuern, benötigt man das Paket *javax.microedition.sensor*. Über diese universelle Sensorschnittstelle kann auf beliebige (z.B. Temperatur- oder Beschleunigungs) Sensoren des mobilen Gerätes zugegriffen werden. Damit die Anwendung beim Auslesen der Sensorwerte nicht die restliche Anwendung blockiert, empfiehlt es sich, diese Operationen in einen separaten Thread auszulagern. Entweder werden die vom Sensor-Thread zuvor ausgelesenen Daten vom Anwendungs-Thread über eine synchrone Methode

abgefragt oder die Anwendung wird als eine Art Observer asynchron vom Sensor-Thread benachrichtigt.

Der Acceleration-Sensor-Prototyp, der zu Beginn der Bachelorarbeit erstellt wurde, zeigt, dass die Verwendung des Beschleunigungssensors mit einem Aufwand von ungefähr zwei Tagen machbar ist. Die Voraussetzungen sind ein mobiles Gerät mit einem Beschleunigungssensor und die Unterstützung des MIDP 2.1 API mit dem Paket *javax.microedition.sensor* des [24]JSR 256. Der Prototyp wurde mangels Hardware mit MIDP 2.1 Unterstützung nur im Phone Emulator des Wireless Toolkits von Sun Microsystems getestet.

Leider unterstützen aktuelle Modelle wie z.B. das Nokia N95 und das Nokia S60 das Sensor API noch nicht, obwohl sie Beschleunigungssensoren haben. Es ist also nicht möglich bei diesen mobilen Geräten direkt aus einer JavaME-Anwendung an die Sensordaten zu kommen. Die einzige Lösung wäre eine Symbian-Anwendung, welche die Sensordaten ausliest und diese der JavaME-Anwendung über einen Socket zur Verfügung stellt.

2.6.4 Kamera

Der Kamera-Prototyp zeigt, wie über das MMAPI [19] ein Einzelbild und ein Video Stream von der Kamera ausgelesen und verarbeitet werden können. Obwohl der Standard festlegt, unter welchem Namen die Kamera für die Aufnahme angesprochen werden sollte (*capture://audio_video* oder *capture://video*), halten sich einige Hersteller nicht daran und verwenden eigene Gerätenamen (z.B. *capture://devcam0*). Über das API können die verfügbaren Geräte zusammen mit ihren Fähigkeiten aufgelistet werden.

Wenn die Kamera im Video-Modus angesprochen wird, werden die Daten intern zwischengespeichert, was zu einer Verzögerung bei der Übertragung führt. Da das Auslösen eines Einzelbildes etwa 1 Sekunde in Anspruch nimmt und gleichzeitig einen Auslöseton abspielt, ist es nicht möglich bzw. tragbar, Videoaufnahmen in Echtzeit mittels einer Einzelbildfolge zu erstellen.

Die Voraussetzungen für die Aufnahmefunktion der Kamera sind ein MIDP 2.0 kompatibles Gerät mit dem MMAPI in der Version 1.0 oder neuer.

2.6.5 GPS

Um den GPS-Empfänger eines mobilen Gerätes zu verwenden benötigt man das Paket *javax.microedition.location*. Über dieses API können Koordinaten der aktuellen Position ausgelesen werden. Je nach Gerät, Empfangsmöglichkeit und Genauigkeit werden dafür die GPS oder aber auch die Cell-Informationen der Mobilfunkantennen verwendet. Dies ist für den Anwender jedoch transparent.

Der GPS-Prototyp zeigte, dass der Zugriff auf Positionsdaten machbar und mit einem Aufwand von ungefähr einem Tag umsetzbar ist. Voraussetzung dafür ist ein mobiles Gerät, welches über einen internen oder externen GPS-Empfänger verfügt und das MIDP 2.0 API mit dem Paket *javax.microedition.location* des [18]JSR 179 unterstützt.

2.6.6 Cell Info

Im Februar 2009 war noch kein standardisiertes API zum Auslesen der Zellinformationen bzw. des aktiven Base Station Transceivers (BST) verfügbar. Es ist zu erwarten, dass die Hersteller in der nahen Zukunft ein API dafür spezifizieren und implementieren werden, jedoch kann heute nur über ein natives API (z.B. mit Hilfe des C++ SDKs für Symbian) auf die Daten zugegriffen werden.

2.6.7 Vibrationsalarm

Heute haben alle Mobiltelefone eine Vibrationseinheit eingebaut. Der Zugriff für die Aktivierung der Vibrationsfunktion ist im Java API sehr einfach gehalten. Dazu muss lediglich die Methode *vibrate()* auf der Klasse *Display* aufgerufen werden mit der Angabe der Dauer in Millisekunden. Die Vibrations-Funktion wird herstellerübergreifend unterstützt und braucht keine speziellen Konfigurationen.

2.6.8 Property Strings

Der Property String Prototyp sollte zeigen, ob alle, auf einem mobilen Gerät vorhandenen Property Strings, abschliessend ausgelesen und aufgelistet werden können. Dies ist leider nicht möglich. Die Property Strings können nur einzeln mit *System.getProperty(String key)* unter der Angabe eines bestimmten Key Strings ausgelesen werden. Der Key muss also im Vornherein bekannt sein.

Folgende Property Strings sind je nach unterstütztem JSR vorhanden:

Key Strings	Beschreibung
microedition.profiles	Für MIDP 2.0 Geräte muss dieses Property mind. “MIDP-20.0” enthalten
microedition.configuration	Die unterstützte JavaME Konfiguration z.B. “CLDC-1.0”
microedition.locale	Der Name der aktuellen Spracheinstellung z.B. “en-US”
microedition.platform	Der Hostname. Meistens “Hersteller Modellnummer/Softwareversion”
microedition.encoding	Character Encoding z.B. “ISO-8859-1”
microedition.commports	COM Ports Komma getrennt aufgelistet
microedition.hostname	In der Methode getLocalAddress() in MIDP 2.0 javax.microedition.io.SocketConnection definiert.
microedition.jtwi.version	Wenn JTWI unterstützt wird muss “1.0” zurückgegeben werden
microedition.msa.version	Versionsnummer der unterstützten MSA Spezifikation z.B. “1.0” oder “1.0-SUBSET”

Tabelle 2.4: CLDC 1.0 & 1.1, MIDP 1.0 & 2.0, and JTWI 1.0

Key String	Beschreibung

Key String	Beschreibung
microedition.media.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.pim.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.m3g.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.location.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
bluetooth.api.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.io.file .FileConnection.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.global.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.chapi.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.sip.version	Wenn das optionale Paket unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
wireless.messaging.version	Wenn das optionale Paket Wireless Messaging API (JSR-120/205) unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.amms.version	Wenn das optionale Paket Advanced Multimedia Supplements (JSR-234) unterstütz ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.m2g.version	Wenn das optionale Paket Scalable 2D Vector Graphics API for JavaME (JSR-226) unterstütz ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.payment.version	Wenn das optionale Paket Payment API (JSR-229) unterstütz ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.contactless.version	Wenn das optionale Paket Contactless Communication API (JSR-257) unterstütz ist, wird ein Versions String zurückgegeben, sonst *null*
microedition.sensor.version	Wenn das optionale Paket Mobile Sensor API (JSR-256) unterstütz ist, wird ein Versions String zurückgegeben, sonst *null*
obex.api.version	Wenn das optionale Paket Object Exchange Protocol (OBEX) API (JSR-82) unterstützt ist, wird ein Versions String zurückgegeben, sonst *null*

Tabelle 2.5: Package discovery

2.7 Feature-Unterstützung durch JavaME

2.7.1 Hintergrundbeleuchtung

Die Steuerung der Hintergrundbeleuchtung des Displays auf mobilen Geräten wird über das JavaME API nicht angeboten. Die einzige standardisierte Funktion im API, welche die Hintergrundbeleuchtung beeinflusst, ist *Display.flashBacklight(duration)*. Mit dieser kann die Hintergrundbeleuchtung jedoch nur zum Blinken gebracht werden.

Funktionen für die direkte Steuerung der Hintergrundbeleuchtung sind herstellerspezifisch und alle unterschiedlich. Im Folgenden sind drei Beispiele von bekannten Herstellern aufgelistet.

Nokia

num bezeichnet die Nummer des Lichtes. Für die Hintergrundbeleuchtung des Hauptbildschirms kann immer 0 verwendet werden.

level bezeichnet die Lichtintensität. 0 steht für Licht aus und 100 für volle Leuchtkraft.

```
import com.nokia.mid.ui.*;
DeviceControl.setLights(int num, int level);
```

Samsung

duration wird in Millisekunden angegeben und kann für maximal 10 Sekunden aktiviert werden.

```
LCDLight.on(int duration);
```

Sharp

```
DeviceControl dc = DeviceControl.getDefaultDeviceControl();
dc.setDeviceActive( DeviceControl.BACK_LIGHT, true );
```

2.8 Style Guidelines für JavaME

Für JavaME Anwendungen gibt es keine herstellerübergreifenden Style Guidelines; auch nicht von Sun[29] selbst.

Hersteller von mobilen Geräten bieten Style Guidelines für JavaME Anwendungen an. Meistens haben aber sogar verschiedene Gerätegruppen desselben Herstellers unterschiedliche Style Guidelines. Für die JTourLive Anwendung können also keine vorhandenen Style Guidelines verwendet werden, da die Anwendung herstellerunabhängig sein sollte. Die graphische Oberfläche der JTourLive Anwendung orientiert sich an anderen herstellerunabhängigen JavaME Anwendungen wie zum Beispiel die Fahrplan Software NaviGo[27] der SBB.

2.9 API Zugriffsrechte

Damit JavaME Anwendungen auf Schnittstellen wie zum Beispiel Positionierung (GPS) oder Multimedia (Kamera und Mikrofon) zugreifen dürfen, müssen die dafür notwendigen Zugriffsrechte vorhanden sein. Diese Zugriffsrechte haben je nach Zuteilung zur Untrusted oder Trusted 3rd Party Domain unterschiedliche Konfigurationsmöglichkeiten und Standardeinstellungen. Wenn eine JavaME Anwendung nicht signiert ist, wird sie automatisch zur Untrusted 3rd Party Domain zugeteilt. Wenn sie jedoch durch eine Zertifizierungsstelle signiert wurde und das Zertifikat dieser Zertifizierungsstelle auf dem mobilen Gerät vorhanden ist, wird sie zur Trusted 3rd Party Domain zugeteilt.

Damit das mobile Gerät nicht bei jedem Zugriff auf eine abgesicherte Schnittstelle durch die JavaME Anwendung, den Benutzer um Erlaubnis bittet, muss die Anwendung zwingend signiert sein. Ansonsten kann, wie unter 2.9.1 ersichtlich ist, die Option *always allowed* gar nicht eingestellt werden.

2.9.1 MIDP API access rights

In den folgenden Tabellen sind die Konfigurationsmöglichkeiten und Standardeinstellungen der Zugriffsrechte des MIDP API der aktuellen Versionen 2.0 und 2.1 für die Untrusted 3rd Party Domain (unsigniert) und die Trusted 3rd Party Domain (signiert) aufgelistet. Es gibt noch zwei weitere Domains, die Operator Domain (für Mobilfunkbetreiber) und die Manufacturer Domain (für Hersteller von mobilen Geräten), welche für selbst entwickelte JavaME Anwendungen jedoch nicht in Frage kommen.

Die Tabelleneinträge haben folgende Bedeutung:

yes	auf dem mobilen Gerät einstellbares Zugriffsrecht
default	nach der Installation eingestelltes Zugriffsrecht
-	nicht einstellbar
not specified	es ist dem Hersteller überlassen, welches Zugriffsrecht einstellbar ist und welches als default voreingestellt ist.

Untrusted 3rd Party Domain[26]

MIDP 2.0

MIDP 2.1

Trusted 3rd Party Domain[26]

MIDP 2.0

MIDP 2.1

	No access	Ask always	Ask first time	Always allowed
Network access	yes	default	yes	-
Messaging	yes	default	-	-
App auto start	yes	yes	default	-
Connectivity	yes	-	default	yes
Multimedia	yes	default	yes	-
Read user data	default	-	-	-
Edit user data	default	-	-	-
Positioning	not specified	not specified	not specified	not specified
Landmarks	not specified	not specified	not specified	not specified
Authentication	not specified	not specified	not specified	not specified
Smart Card access	not specified	not specified	not specified	not specified
Phone call	yes	default	-	-
Low level net access	not specified	not specified	not specified	not specified
Restricted messaging	not specified	not specified	not specified	not specified
Call control	not specified	not specified	not specified	not specified

Tabelle 2.6: MIDP 2.0 Zugriffsrechte für unsignierte Anwendungen

	No access	Ask always	Ask first time	Always allowed
Network access	yes	default	yes	-
Messaging	yes	default	-	-
App auto start	yes	default	yes	-
Connectivity	yes	default	yes	yes
Multimedia	yes	default	yes	-
Read user data	yes	default	-	-
Edit user data	yes	default	-	-
Positioning	yes	default	yes	-
Landmarks	yes	default	yes	-
Authentication	default	-	-	-
Smart Card access	default	-	-	-
Phone call	yes	default	-	-
Low level net access	yes	default	yes	-
Restricted messaging	yes	default	-	-
Call control	yes	default	-	-

Tabelle 2.7: MIDP 2.1 Zugriffsrechte für unsignierte Anwendungen

	No access	Ask always	Ask first time	Always allowed
Network access	yes	yes	default	yes
Messaging	yes	default	-	-
App auto start	yes	yes	default	yes
Connectivity	yes	-	default	yes
Multimedia	yes	-	default	yes
Read user data	yes	default	yes	yes
Edit user data	yes	default	yes	yes
Positioning	not specified	not specified	not specified	not specified
Landmarks	not specified	not specified	not specified	not specified
Authentication	not specified	not specified	not specified	not specifiedes
Smart Card access	not specified	not specified	not specified	not specified
Phone call	yes	default	-	-
Low level net access	not specified	not specified	not specified	not specified
Restricted messaging	not specified	not specified	not specified	not specified
Call control	not specified	not specified	not specified	not specified

Tabelle 2.8: MIDP 2.0 Zugriffsrechte für signierte Anwendungen

	No access	Ask always	Ask first time	Always allowed
Network access	yes	yes	default	yes
Messaging	yes	default	yes	yes
App auto start	yes	default	yes	yes
Connectivity	yes	yes	default	yes
Multimedia	yes	yes	default	yes
Read user data	yes	default	yes	yes
Edit user data	yes	default	yes	yes
Positioning	yes	yes	default	yes
Landmarks	yes	yes	default	yes
Authentication	yes	yes	default	yes
Smart Card access	yes	yes	default	yes
Phone call	yes	default	yes	yes
Low level net access	yes	yes	default	yes
Restricted messaging	yes	default	yes	yes
Call control	yes	default	yes	yes

Tabelle 2.9: MIDP 2.1 Zugriffsrechte für signierte Anwendungen

2.10 Zertifikate für die Trusted 3rd Party Domain

Um eine eigene JavaME Anwendung auf einem mobilen Gerät unter der Trusted 3rd Party Domain zu installieren, muss diese signiert werden. Am einfachsten ist es, wenn man die Anwendung von einer der folgenden drei Zertifizierstellen signieren lässt, da von diesen die Root-Zertifikate auf den mobilen Geräten der bekanntesten Hersteller[4] vorinstalliert sind.

- Java Verified www.javaverified.com/ql_test_providers.jsp
  Java Verified geht am weitesten in dem der Code auf einem bestimmten mobilen Gerät getestet wird und für dieses als Java Verified signiert wird. Gleichzeitig schützt dieses Zertifikat auch die Integrität des Codes und identifiziert den Herausgeber eindeutig. Die Kosten sind vom Java Verified autorisierten Test Provider abhängig. Die Test Provider sind unter www.javaverified.com/ql_test_providers.jsp aufgeführt.
- Thawte www.thawte.com/code-signing/index.html
  Thawte stellt Software Zertifikate mit einer Gültigkeitsdauer von einem (US$299.00[5]) oder zwei (US$549.00) Jahre aus. Dieses Zertifikat schützt die Integrität der Software und identifiziert den Herausgeber eindeutig.
- Verisign www.verisign.com/code-signing/content-signing-certificates/index.html
  Verisign stellt Software Zertifikate mit einer Gültigkeitsdauer von einem (US$499.00), zwei (US$895.00) oder drei (US$1'295.00) Jahre aus. Dieses Zertifikat schützt die Integrität der Software und identifiziert den Herausgeber eindeutig.

JTourLive wird im Rahmen dieser Bachelorarbeit nicht signiert. Die Anwendung erreicht noch keine Marktreife und ist eher als ausführlichen Prototypen zu betrachten.

2.11 Schnittstellen und Features

2.11.1 Datenquellen und -senken

In der Abbildung 2.3 auf der nächsten Seite ist der System Context mit allen möglichen Datenquellen und -senken zu sehen, mit denen die JTourLive Anwendung kommunizieren könnte. Welche Datenquellen und -senken schlussendlich zum Einsatz kommen, ist im Kapitel 3 auf Seite 25 ersichtlich.

2.11.2 Feature List

In der Feature Liste ist aufgeführt, welche Features von der bisherigen Symbian TourLive Anwendung unterstützt werden und welche in der neuen JavaME JTourLive Anwendung umgesetzt werden, verglichen mit den Features welche Nokia Sport Tracker unterstützt.

Die mit * gekennzeichneten Werte werden durch die Anwendung berechnet und sind keine rein gemessenen Werte.

[4] Nokia, Samsung, Sony Ericsson, Motorola
[5] Alle Preise gemäss offizieller Webseite, März 2009

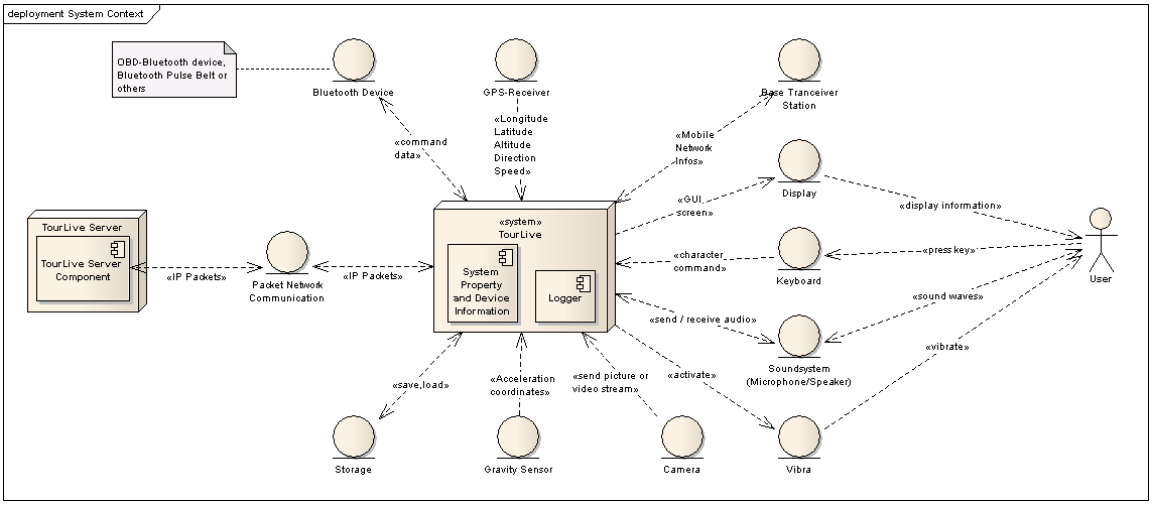

Abbildung 2.3: System Context

Kategorie	Gruppe	Feature	TourLive	JTourLive 09	NokiaSport
(Bluetooth)	GPS	Geschw.	x	x	x
		Höhe	x	x	x
		Richtung	x	x	x
		Steigung*	-	-	-
		Koordinaten	x	x	x
Bluetooth	OBD	Geschw.	x	x	-
		Umdrehungen	x	x	-
		Kühlertemp.	x	x	-
		Gang*	x	x	-
		Gas/Lastwert	x	x	-
		Akt. Verbrauch*	x	x	-
		Total Verbrauch*	x	x	-
		OBD Distanz*	x	x	-
		Mass Air Flow	x	x	-
		VIN	x	x	-
		Feature Set	x	x	-
		Total Distanz*	x	-	-
		Total Verbrauch*	x	-	-
Bluetooth	Puls	Puls	x	x	x
		Puls Schnitt*	x	x	-
		Puls min*	x	x	-
		Puls max*	x	x	-
		Kalorien*	x	x	x

Kategorie	Gruppe	Feature	TourLive	JTourLive 09	NokiaSport
		Akku	x	-	-
		(History / Erfolgskurve)	-	-	x
		Trainings-Vorschläge	-	-	-
Mobiles Gerät	GSM/UMTS Antenne	Zellen-ID	x	-	-
		Area(LAC)	x	-	-
		Signal	x	-	-
		Akku	x	-	-
		Netzwerk	x	-	-
		Netzwerk-ID	x	-	-
	Kamera	Bilder	x	x	-
		Video Stream	x	-	-
	Beschl. Sensor	reine Daten	-	-	-
		Schritt-Zähler*	-	-	x
	Mikrofon	Audio Stream	-	-	-
	Aggregierte Daten	Zeit Total	x	-	-
		Distanz Total*	x	-	-
		Höhe Total*	x	-	-
		Zeit Tour	x	x	x
		Distanz Tour*	x	x	x
		Höhe Tour*	x	x	x
		Zeit Trip	x	-	-
		Distanz Trip*	x	x	x
		Höhe Trip*	x	-	-
		D.Geschw.*	x	-	x
		Zeit	x	x	x
		Datum	x	x	x

Tabelle 2.10: Features der TourLive-Anwendung

2.12 Automatische Aktualisierung von JavaME-Anwendungen

Seit MIDP 2.0 ist es möglich durch die Methode *javax.microedition.midlet.MIDlet.platformRequest()* einen einfachen Befehl ans Mobiltelefon zu senden. Dieser Befehl kann z.B. ein Anruf oder ein Download sein und wird in der Form einer URL übergeben. Falls die URL zu einer .JAD-Datei führt, muss das Mobiltelefon laut Spezifikation dem Benutzer den Download der Anwendung anbieten, wodurch ein sehr einfacher Updatemechanismus für JavaME-Anwendungen möglich wird.

Denkbar sind hierbei verschiedene Stufen von Updates, die zu unterschiedlichen Zeiten überprüft werden:

Normales Update Dieses Update beinhaltet nur "kosmetische" Änderungen. Im Online-Modus wird das Update automatisch angeboten, jedoch wird der Benutzer nur einmalig darauf aufmerksam gemacht. Der Benutzer kann, falls er JTourLive im Offline-Modus verwendet, oder den Download abgebrochen hat, diesen jederzeit über das Hauptmenü starten.

Wichtiges Update Dieses Update behebt wichtige Fehler, die die normale Ausführung stark stören oder Sicherheitslöcher öffnen. Der Benutzer wird jedes mal vor der Verwendung des Online-Modus gefragt, ob er das Update machen will.

Änderung des APIs Falls sich etwas am Online-API ändert, ist es für die Anwendung nicht mehr möglich, mit der alten Version fortzufahren. Der Benutzer hat die Wahl im Offline-Modus zu arbeiten, oder das Update einzuspielen.

Das API zur Überprüfung der Version muss die Möglichkeit haben, neben der Dringlichkeit auch eine Nachricht zu übermitteln, die dem Benutzer angezeigt wird. Diese soll dem Benutzer helfen können über das Update zu entscheiden. Um die Anzahl Requests möglichst klein zu halten, sollte der API-Aufruf zur Erstellung eines Trips auch prüfen können, ob die Version aktuell ist.

Die gewählte Lösung bietet keine Benachrichtigung bei der Verwendung des Online-Modus, ausser wenn das API geändert hat. In diesem Falle bricht der Online-Modus mit einer Fehlermeldung ab.

3 Realisierung

3.1 Java Micro Edition (JavaME)

Praktisch alle neuen Geräte der grössten Mobiltelefon-Hersteller[1] unterstützen Java Micro Edition (JavaME). Dank dieser Verbreitung von JavaME sollten in Java programmierte Anwendungen auf sehr vielen Mobiltelefonen lauffähig sein. Da JavaME eine Untermenge von Java Standard Edition (JavaSE) ist, können JavaME-Anwendungen, abgesehen von speziellen MIDlet-Klassen, auch unter JavaSE, d.h. auf beliebigen Computern in der Java Virtual Machine (JVM) ausgeführt werden. Nicht unterstützt wird JavaME auf dem Apple iPhone.

3.1.1 Einschränkungen

Unter JavaME sind viele selbstverständliche Funktionalitäten aus JavaSE nicht oder nur eingeschränkt vorhanden. Die wichtigsten Einschränkungen sind hier kurz aufgeführt:

- keine Fliesskommaarithmetik (mit CLDC 1.1 wurde dies geändert)
- Eingeschränkte Fehlerbehandlung (Exception-Handling ist vorhanden, aber nur eingeschränkte Error-Klassen verfügbar)
- Kein Java Native Interface (JNI)
- Keine benutzerdefinierten Classloader
- Keine Reflection
- Keine Thread-Gruppen
- Keine Finalisierung (kein object.finalize())
- Keine Weak References

3.1.2 MIDlet

Ein MIDlet ist eine Software Komponente, welche in Java geschrieben ist und auf der JVM von mobilen Geräten wie Mobiltelefonen und PDAs lauffähig ist. Anders als ein JavaSE Programm ist ein MIDlet in Konfiguration und Profil aufgeteilt.

[1] Nokia, Samsung, Sony Ericsson, Motorola und LG

Connected Limited Device Configuration (CLDC)

Die Connected Limited Device Configuration (CLDC) spezifiziert die Mindestanforderungen an die Hardware des mobilen Gerätes. Dazu gehören Speicher-, Display-, Eingabe- und Netzwerkvoraussetzungen, sowie die Eigenschaften der JVM und die mitgelieferten Java-Bibliotheken.

Mobile Information Device Profile (MIDP)

Das Mobile Information Device Profile (MIDP) legt ein Java-Applet ähnliches Anwendungs-Modell fest, welches auf mobilen Endgeräten läuft. Ein MIDlet hat eingeschränkte Rechte und durchläuft verschiedene Phasen in seinem Lebenszyklus. Die Phasenübergänge werden über ein vordefiniertes API vom mobilen Gerät initiiert.

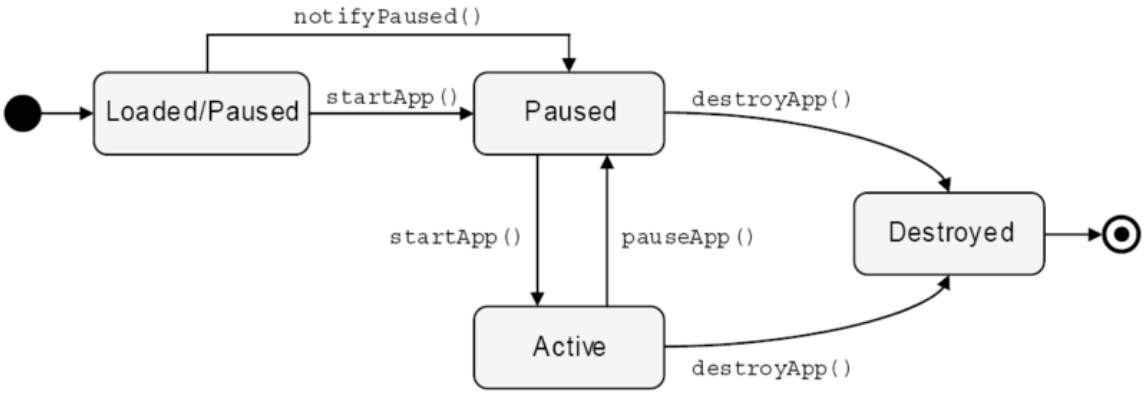

Abbildung 3.1: Lebenszyklus eines MIDlets [34]

3.1.3 Deployment

Bevor ein Java Code ausgeliefert werden kann, werden alle Java-Klassen verifiziert. Alle verifizierten Klassen und Packages werden nach dem JAR-Standard von JavaSE in ein herkömmliches JAR-File gepackt. Dieses wird dann auf das mobile Gerät deployed. Optional kann zusätzlich noch ein Java Application Descriptor (JAD) mitgeliefert werden, der Metadaten über das JAR-File enthält. Enthaltene Informationen sind z.B. Name, Herausgeber, Version, Grösse, URL des JAR-Files und benötigte Bibliotheken oder Pakete. Dies ist für mobile Geräte von grosser Bedeutung, da die Ressourcen und die Bandbreite oft stark eingeschränkt sind und so bereits vor dem Software-Download festgestellt werden kann, ob z.B. das Gerät über genügend freien Speicher verfügt.

3.2 Aufbau von JTourLive

Die JTourLive Anwendung wurde in drei Projekte aufgeteilt, um die unterschiedlichen Java APIs klar zu trennen. So enthält das TourLiveMobile Projekt alle JavaME spezifischen In-

terfaces und Klassen. Dazu zählen vor allem die MIDlet spezifischen GUI Klassen. Das TourLiveDesktop Projekt ist JavaSE spezifisch und enthält vor allem die Swing GUI Klassen. Das TourLiveCore Projekt muss sowohl unter JavaME wie auch unter JavaSE kompilieren. Es definiert alle wichtigen Interfaces auf die aus den TourLiveDesktop und TourLiveMobile Projekten zugegriffen wird.

3.2.1 TourLiveCore

Im TourLiveCore Projekt befindet sich die Engine Klasse, die das Kernstück der gesamten JTourLive Anwendung darstellt. Bei der Engine werden alle benötigten Provider, Sinks und Screens registriert. Sie gibt auch den Takt an, mit welchem Daten aus den Providern gelesen und in die Sinks geschrieben werden.

3.2.2 TourLiveDesktop

TourLiveDesktop ist das Projekt, welches alle Swing GUI spezifischen Pakete enthält. Es werden wieder die gleichen Interfaces implementiert wie im TourLiveMobile Projekt, einfach mit Swing GUI-Komponenten. Anstelle der MIDlet GUI-Komponenten.

3.2.3 TourLiveMobile

Das TourLiveMobile Projekt enthält als wichtigste Klasse das TourLiveMidlet MIDlet. Dieses MIDlet ist der Startpunkt der JTourLive Anwendung auf mobilen Geräten. Diese Klasse registriert alle benötigten Provider, Sinks und Screens beim Core.

3.3 Core

3.3.1 Engine

Die *Engine* ist die zentrale Klasse des JTourLive Kerns. Bei der *Engine* werden alle *DataSink, DataProvider* und *ScreenProvider* registriert. Die *Engine* verwaltet die Konfigurationen für alle Module. Die Engine verwaltet auch den *EngineThread* und steuert somit den Start und das Ende einer Aufnahme. Die *Engine* bietet ausserdem Funktionen zur Behandlung von Ausnahmen und einige wichtige zentrale Einstellungen an und verwaltet die Konfiguration der gesamten Applikation.

3.3.2 EngineThread

Der *EngineThread* ist dafür zuständig die Aufnahme durchzuführen. Er kennt alle aktiven *DataSink* und *DataProvider*, liest im Takt die Datenpunkte aus und gibt diese an die Senken weiter zum speichern. Der *EngineThread* sendet alle neuen Datenpunkte auch an das GUI weiter und gibt durch die Methode *tick()* auch den Taktzyklus an.

Durch seine Funktion wir dieser Thread zum wichtigsten Element einer Aufnahme. So wird hier das gesamte Threading und die Synchronisierungsarbeit geleistet. Über das Interface

EngineState können ausserdem wichtige Informationen zum aktuellen Zustand der Engine und der laufenden Aufnahme abgefragt werden.

3.3.3 ConcreteDataPoint

Die *ConcreteDataPoint* Klasse ist eine Art Behälter Klasse von JTourLive. Sie nimmt von allen *DataProvider* die Daten auf und übergibt sie der *DataSink*. Die *ConcreteDataPoint* Klasse implementiert 2 Interfaces. Einerseits das *DataPointWriter* Interface, welches von den *DataProvider* für den Schreibzugriff verwendet wird. Andererseits das *DataPointReader* Interface, welches von der *DataSink* für den Lesezugriff verwendet wird.

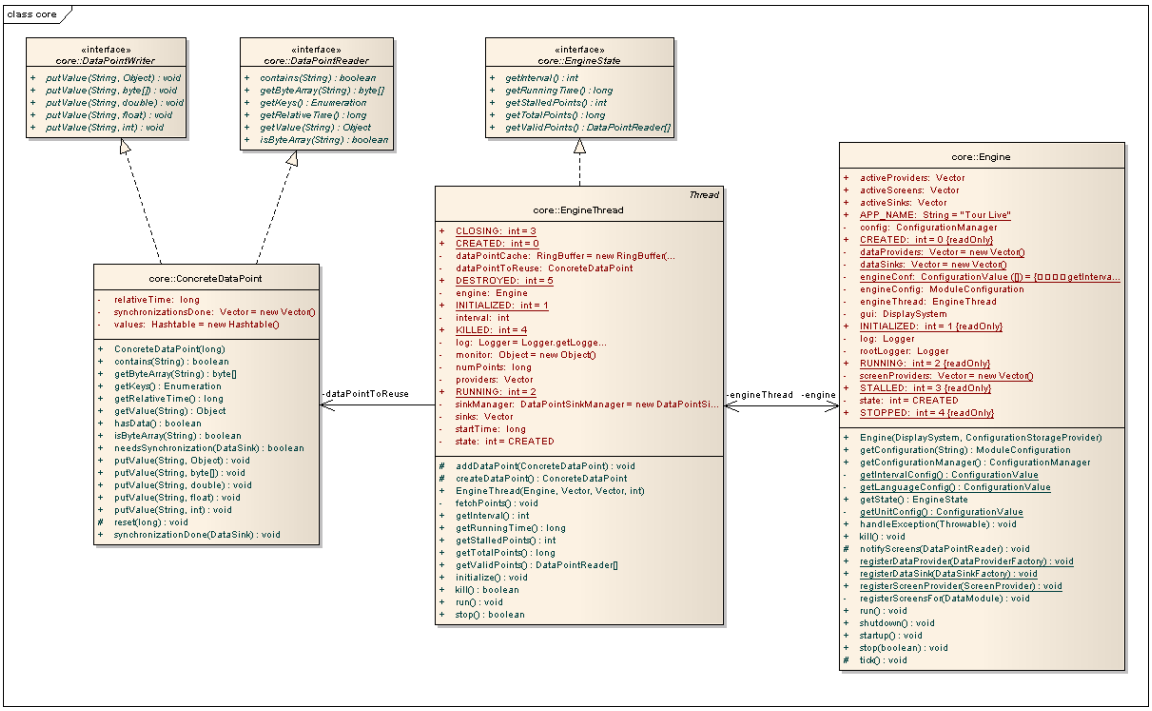

Abbildung 3.2: ch.hsr.tourlive.core

3.4 Sinks

Das *DataSink* Interface wird von allen Daten-Senken implementiert um Daten aufzunehmen. *DataPointSink* ist eine Spezialisierung von *DataSink* um *ConcreteDataPoint* als *DataPointReader* entgegen zu nehmen. Das *DataSinkFactory* Interface ist das gemeinsame Interface von allen Factories, welche eine konkrete Instanz einer *DataSink* zurück liefern.

3.5 Provider

Analog zu den Sinks im Abschnitt 3.4 gibt es auch für alle Daten Provider ein gemeinsames Interface *DataProvider* und eine Spezialisierung *DataPointProvider* davon, welche *Concre-*

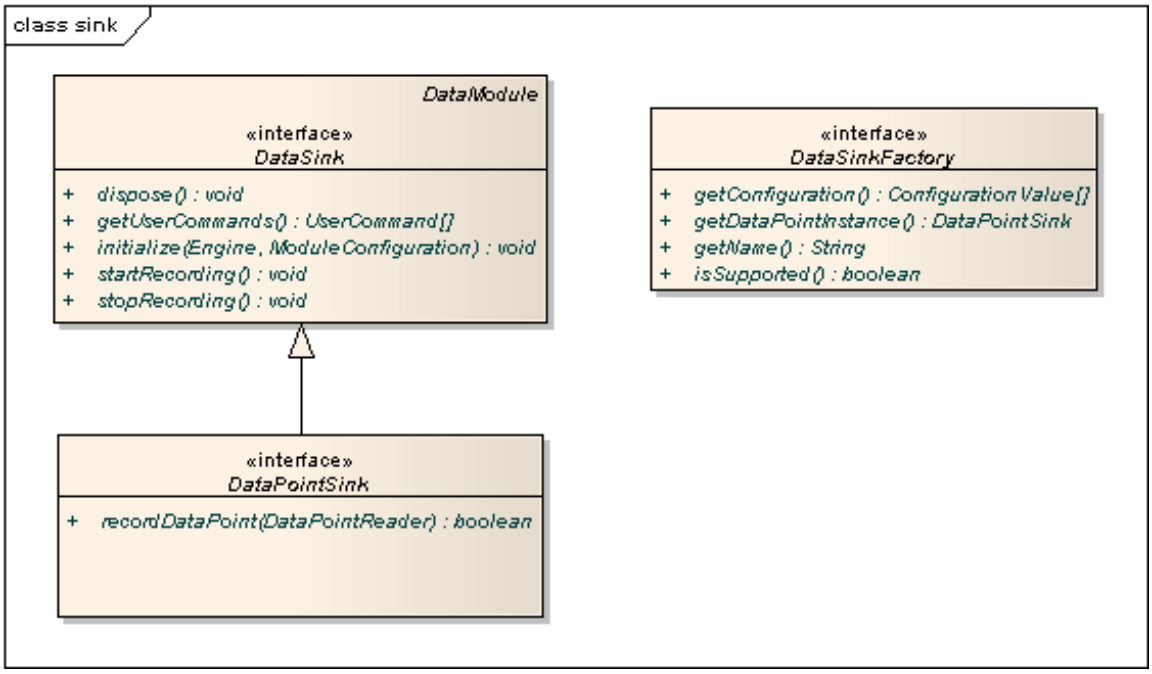

Abbildung 3.3: ch.hsr.tourlive.sinks

*teDataPoint*s in der Form von *DataPointWriter*n mit Daten füllt. *DataProviderFactory* ist wieder ein gemeinsames Interface für alle Factories, welche Instanzen von *DataProvider*n zurückgeben können.

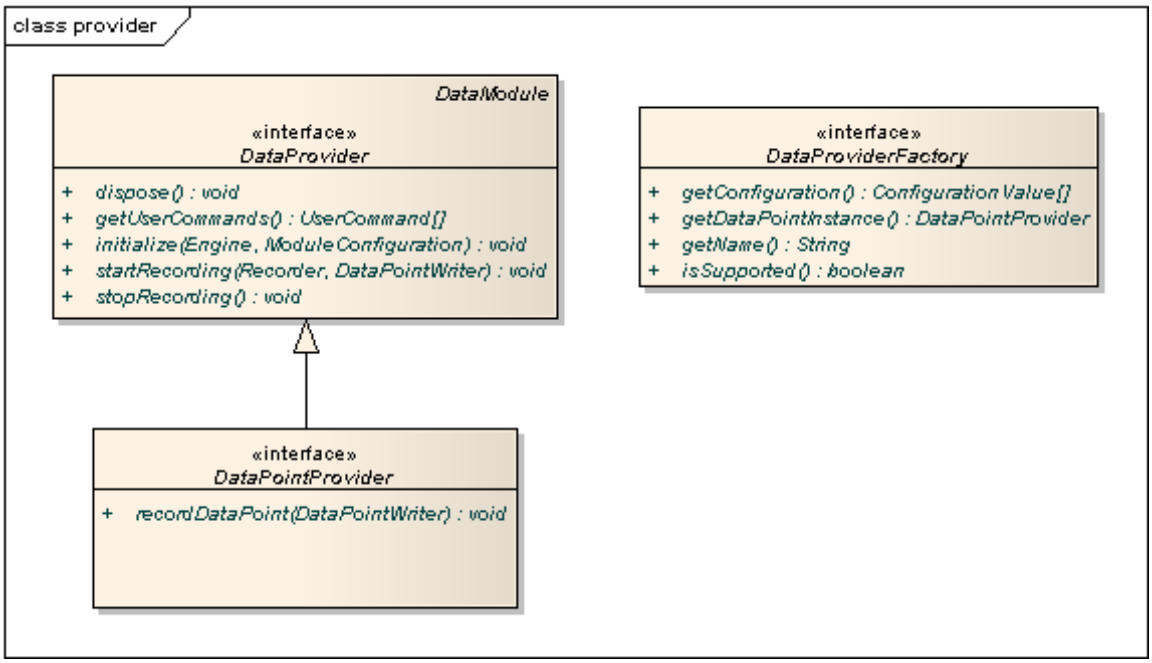

Abbildung 3.4: ch.hsr.tourlive.provider

3.5.1 OBD Provider

Die *OBDDataProviderFactory* implementiert das *DataProviderFactory* Interface und gibt eine Instanz des *OBDDataProvider*s zurück. Der *OBDDataProvider* erstellt über eine *OBDConnection* eine Verbindung zu einem Bluetooth-OBD-II-Adapter und erzeugt anschliessend eine Instanz des *OBDReader*. Der *OBDReader* enthält einen *OBDReaderThread*, welcher im Hintergrund läuft und permanent Daten aus der OBD-II-Schnittstelle des Fahrzeuges abfragt. Für die Abfragen der verschiedenen Werte wiederum verwendet der *OBDReaderThread OBDMessage*s, welche die verschiedenen OBD-II-Befehle abstrahieren. Um die erhaltenen Antworten von der OBD-II-Schnittstelle einfach parsen zu können, wird ein mit JFlex[16] generierter Lexer, der *OBDLexer,* verwendet. Für die Erkennung der gefahrenen Gänge schlussendlich wird die *OBDCalculator* Klasse eingesetzt. Der *OBDCalculator* erkennt die Gänge während dem Fahren anhand der Drehzahl und der Geschwindigkeit und speichert die Übersetzungsverhältnisse mit der VIN auf dem Aufzeichnungsgerät ab, damit die Zeit für die Erkennung nur beim allerersten Einsatz mit einem Fahrzeug in Kauf genommen werden muss.

3.5.2 GPS Provider

Die *GPSDataProviderFactory* implementiert das *DataProviderFactor* Interface und liefert eine Instanz der *GPSDataProvider* Klasse zurück. Der *GPSDataProvider* implementiert das *DataPointProvider* Interface und liefert die Daten aus dem GPS-Modul. Das GPS-Modul wird von der *GPS* Klasse abstrahiert. Um die GPS-Höhe vom WGS84-Format in die effektive Höhe über Meer zu korrigieren wird die *AltitudeCorrector* Klasse verwendet. Details zur Höhenkorrektur werden im Abschnitt 2.1.1 auf Seite 6 genauer erläutert.

3.5.3 Puls Provider

Die *PulseDataProviderFactory* implementiert das *DataProviderFactory* Interface und liefert eine Instanz der *PulseDataProvider* Klasse zurück. Der *PulseDataProvider* baut mit Hilfe der *PulseConnection* eine Verbindung zum Bluetooth-Pulsgurt auf. Anschliessend wird eine Instanz der *PulseReader* Klasse erstellt, welche die Kommunikation mit dem Pulsgurt aufnimmt und die Herzfrequenz ausliest. Im *PulseReader* werden auch die Berechnungen vom Maximal-Puls, Minimal-Puls, Durchschnitts-Puls und der verbrauchten Kalorien gemacht.

3.5.4 Image Provider

Die *ImageDataProviderFactory* implementiert das *DataProviderFactory* Interface und liefert eine Instanz der *ImageDataProvider* Klasse zurück. Der *ImageDataProvider* erstellt im manuellen Modus auf einen Benutzerbefehl hin oder im automatischen Modus periodisch ein Bild über die im Aufzeichnungsgerät integrierte Kamera und schreibt das Bild in einen *ConcreteDataPoint.*

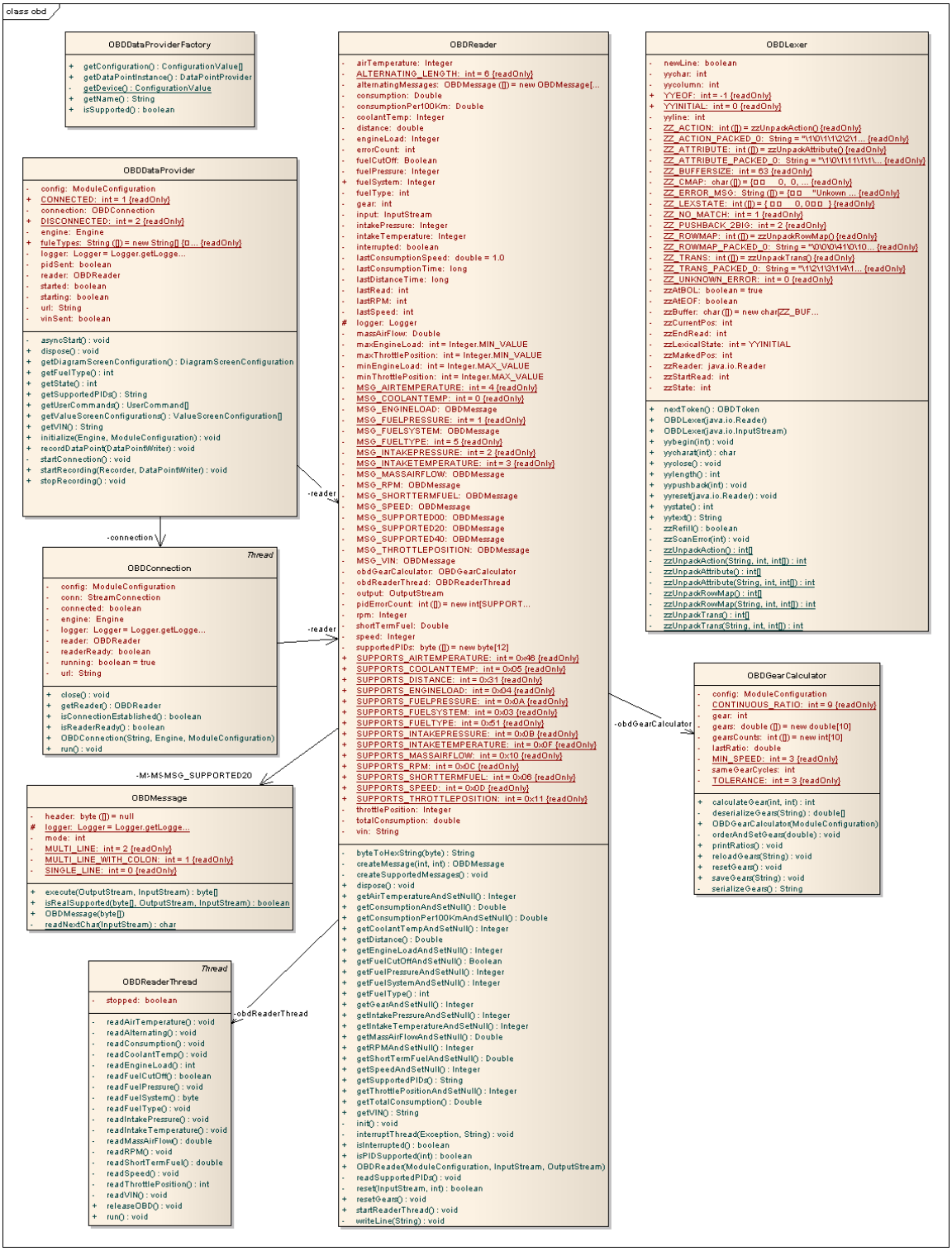

Abbildung 3.5: ch.hsr.tourlive.provider.obd

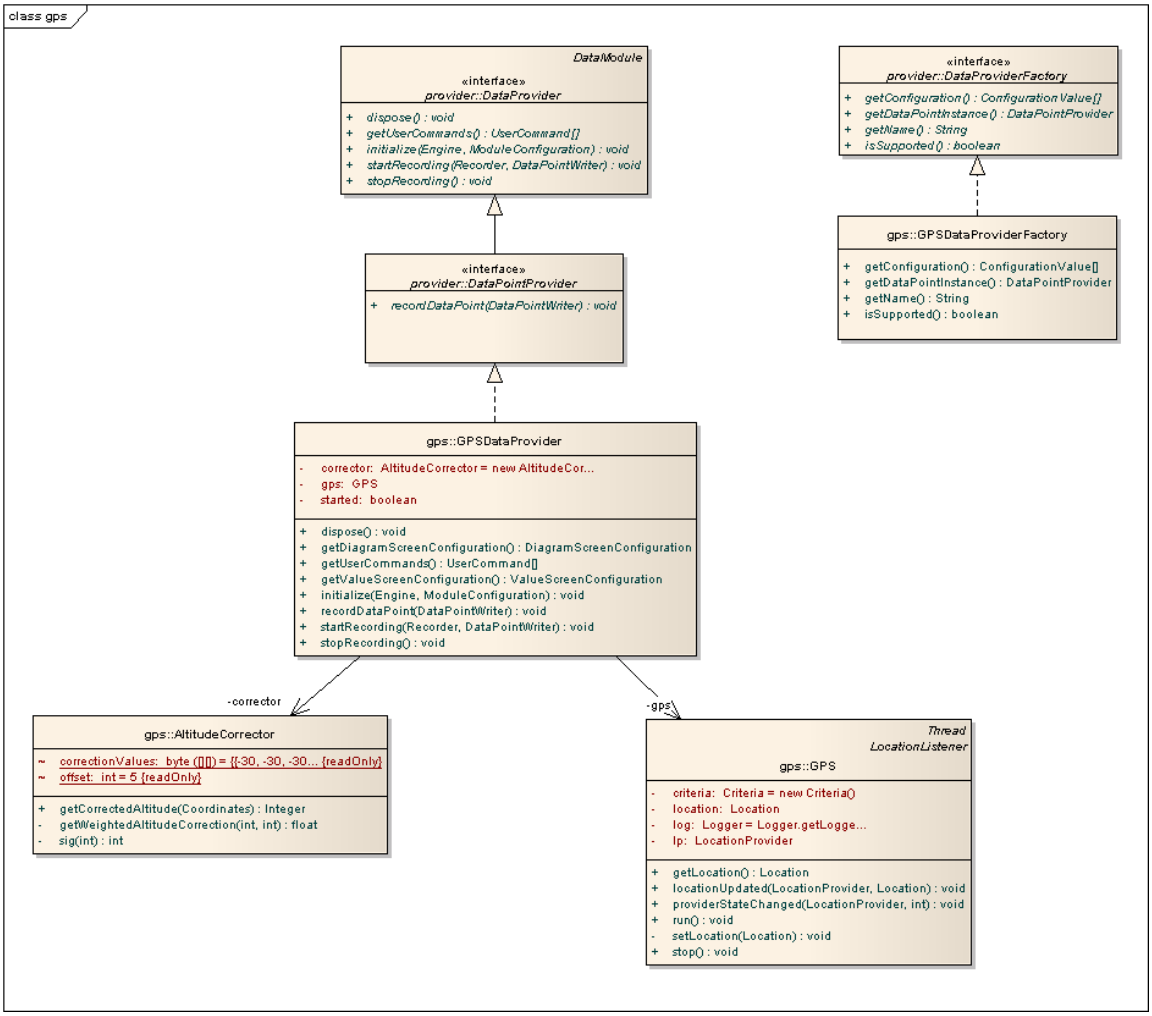

Abbildung 3.6: ch.hsr.tourlive.provider.gps

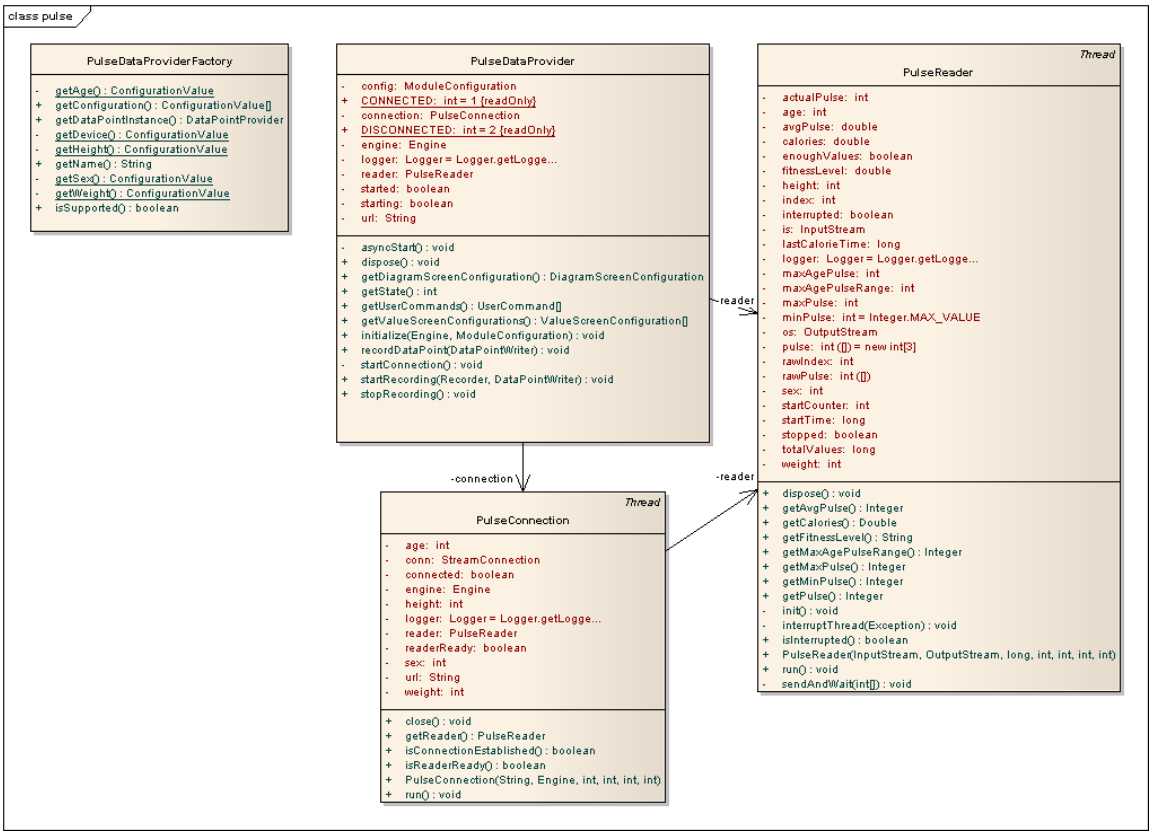

Abbildung 3.7: ch.hsr.tourlive.provider.pulse

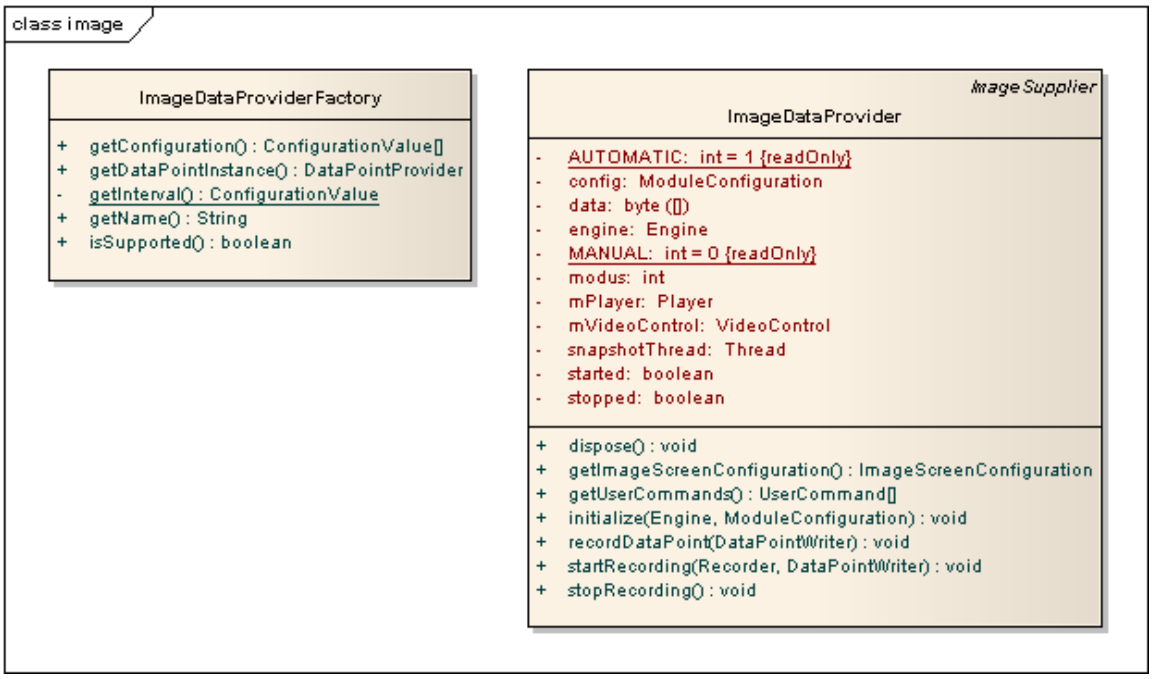

Abbildung 3.8: ch.hsr.tourlive.provider.image

3.6 MIDlet GUI

Das MIDlet GUI ist speziell auf kleine Bildschirme zugeschnitten. So werden z.B. einzelne GUI-Komponenten auf verschiedenen Geräten unterschiedlich dargestellt um den Platz optimal auszunutzen. Das Design und der Ablauf des MIDlet GUI orientiert sich am GUI-Entwurf in Abbildung 3.9 auf der nächsten Seite

3.7 Swing GUI

Das Swing GUI ist eine Implementation von TourLive für den Desktop. Neben der Erweiterung der unterstützten Plattformen zeigt die Implementation dieses GUIs auch, dass das Design des Kerns und der Interfaces korrekt ist und die Aufteilung zwischen Core und UI richtig vorgenommen wurde.

Die Desktop Variante ist nicht so mächtig wie das MIDlet GUI, da das Hauptaugenmerk der Arbeit auf den Mobilen Plattformen liegt. So bietet die Desktop-Variante keine Unterstützung für Kameras und bietet noch keine Mehrsprachigkeit. Die Implementation ist also mehr als Machbarkeitsstudie und Grundstock für eine Weiterentwicklung anzusehen.

3.8 Externe Klassen und Libraries

3.8.1 net.sf.microlog (MIDlet Client)

Microlog[17] ist eine OpenSource Logging-Komponente, die speziell für die Verwendung in JavaME- und Android-Anwendungen entwickelt wurde. Das API wurde zur einfachen Austauschbarkeit vom bekannten und oft verwendeten Projekt Log4J[2] übernommen.

Microlog überzeugt durch eine einfache Verwendung, vielen Features und eine kleine Codebasis. Microlog verwendet die Apache Lizenz in der Version 2.0[1], welche eine freie Nutzung auch für kommerzielle Zwecke erlaubt. Die heruntergeladenen Quelltexte wurden ohne Änderung übernommen.

3.8.2 net.sourceforge.jmicropolygon (MIDlet Client)

JMicroPolygon[32] ist eine OpenSource Grafik-Bibliothek, die einige Funktionen zur einfachen Darstellung von komplexen Polygonen anbietet. Das Projekt wird nicht mehr aktiv weiterentwickelt, hat jedoch keine bekannten Fehler und erledigt seine Aufgabe sehr performant.

JMicroPolygon steht unter der Apache Lizenz in der Version 2.0[1] und der Lesser GPL[11], welche beide eine freie Nutzung auch für kommerzielle Zwecke erlaubt. Die heruntergeladenen Quelltexte wurden ohne Änderung übernommen.

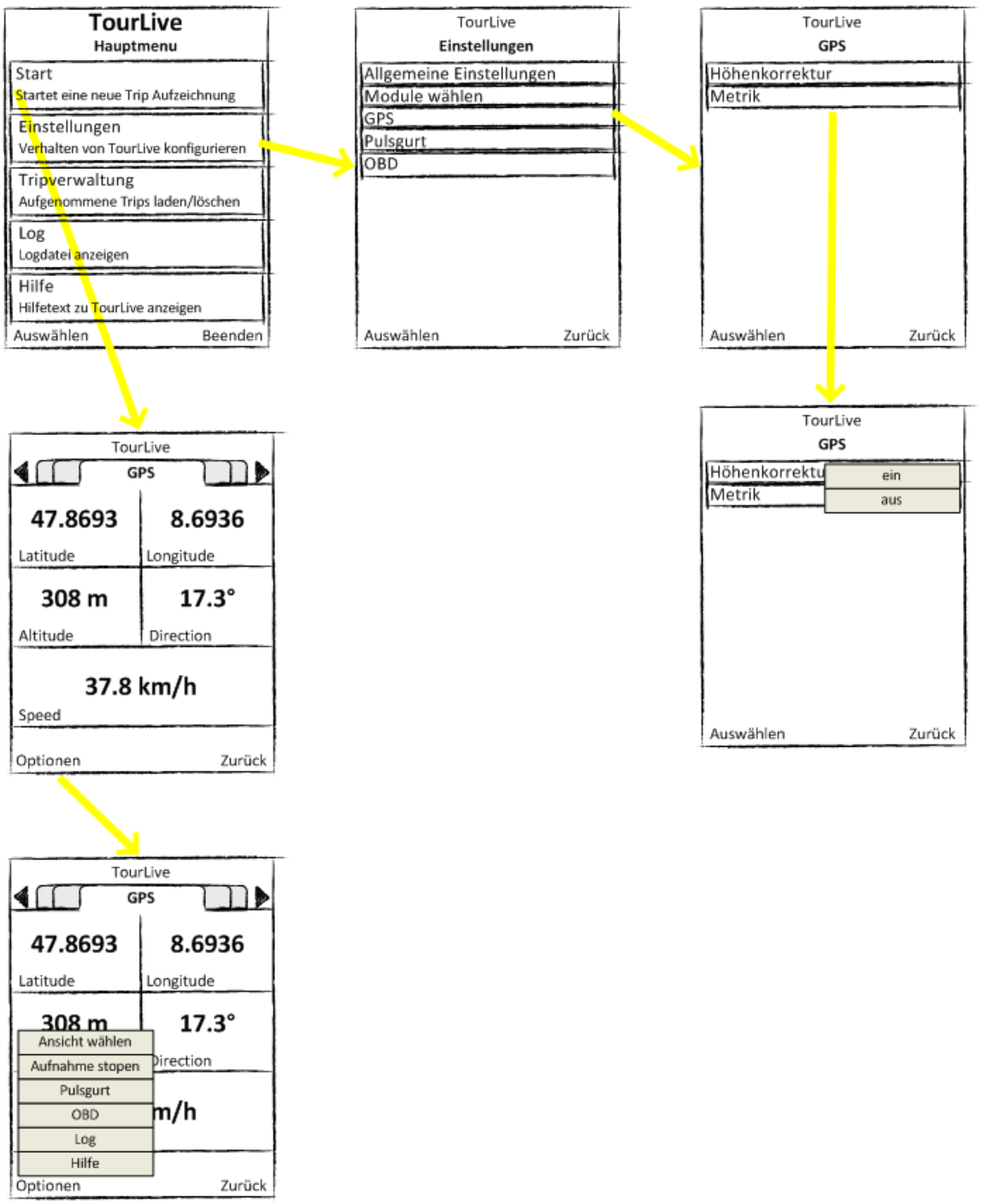

Abbildung 3.9: MIDlet GUI Entwurf

3.8.3 BlueCove (Desktop Client)

BlueCove[31] ist eine freie Implementation des JSR-82 Bluetooth Stacks für JavaSE. BlueCove bietet Implementationen für verschiedene native Bluetooth-Stacks und ist so plattformunabhängig[2].

BlueCove steht unter der Apache Lizenz in der Version 2.0[1]. Die optionale Implementation für den BlueZ-Stack für Linux steht unter der GPL[12]. Die heruntergeladenen Binärpakete wurden ohne Änderungen eingebunden.

3.8.4 Apache Commons Codec, HttpClient und Logging (Desktop Client)

Apache Commons HttpClient[4] ist eine freie Implementation des HTTP Standards für Requests zu HTTP Servern. HttpClient bietet ein einfach zu verwendendes und performantes API um GET- und POST-Requests mit Dateien und sonstigen Daten zu senden.

Die kürzliche veröffentlichte Version 4.0 hat ein komplett überarbeitetes API, welches zur Zeit der Verwendung noch sehr knapp dokumentiert war. Aus diesem Grund wird die Version 3.1 verwendet, zu welcher sich viele Tutorials und eine grosse Menge Dokumentation finden lässt.

Apache Commons HttpClient hat Abhängigkeiten zu den Projekten Apache Commons Codec[36] und Apache Commons Logging[7]. Codec bietet Methoden zur Codierung und Decodierung von Strings mit verschiedenen Algorithmen wie Base64 und UUEncode. Logging bietet ein standardisiertes API für verschiedene Logging Komponenten. Beide Bibliotheken werden nicht direkt verwendet und sind nur eingebunden, weil sie als Abhängigkeit benötigt werden.

Apache Commons HttpClient, Codec und Logging stehen unter der Apache Lizenz in der Version 2.0[1]. Die heruntergeladenen Binärpakete wurden ohne Änderungen eingebunden.

3.8.5 JFreeChart (Desktop Client)

JFreeChart[14] ist eine freie Bibliothek zum zeichnen von Diagrammen und Graphen. JFreeChart unterstützt durch ein gut strukturiertes API eine breite Anzahl verschiedener Diagrammtypen und Export-Formate.

JFreeChart steht unter der Lesser GPL[11]. Die heruntergeladenen Binärpakete wurden ohne Änderungen eingebunden.

3.8.6 OpenLAPI (Desktop Client)

OpenLAPI[15] ist eine freie Implementation des JSR-179 Location API für JavaSE und ME. OpenLAPI bietet über das standardisierte API einen einfachen Zugriff zu GPS-Geräten die per Bluetooth mit dem Gerät verbunden sind, sowie Simulatoren, die Daten von einem NMEA[3]-Logfile, Google Earth KML oder einer Zufallsquelle liefern.

[2] Unterstützt werden Windows, Linux und Mac OSX

[3] NMEA ist das Standardformat, welches von fast allen GPS-Geräten verwendet wird

OpenLAPI steht unter der Lesser GPL[11]. Die heruntergeladenen Binärpakete wurden ohne Änderungen eingebunden.

3.9 Auslieferung

JTourLive wird für als JAR-Datei ausgeliefert. Für mobile Geräte muss die JAR-Datei *preverified* sein. Auf Wunsch der cnlab AG wurde die Software vorgängig noch *obfuscated*, um eine *De-Kompilierung* zu verhindern. Es ist aber geplant, dass das JTourLive Projekt später unter einer OpenSource-Lizenz frei verfügbar angeboten wird.

4 Schlussfolgerungen

4.1 Einsatz von JTourLive

JTourLive wurde mit verschiedenen Fahrzeugen und verschiedenen Mobiltelefonen erfolgreich getestet.

4.1.1 Erfolgreich getestete Fahrzeuge

- Audi A4 Avant 2.0 TDI 2009
- Mazda 3
- Seat Ibiza 1.6 2008
- Seat Ibiza FR Turbo
- VW Polo 1.4 TDI 2008

4.1.2 Erfolgreich getestete Mobiltelefone

Nokia

- N95 / N95 8GB
- N82
- E71
- E51
- 5800 XpressMusic (mit TouchScreen)

SonyEricsson

- W910i

4.2 Weiterentwicklungen

Im Folgenden sind Ideen für mögliche Weiterentwicklungen aufgeführt und bewertet nach Realisierbarkeit, Aufwand und Wichtigkeit.

- Allgemeine Features
  - Heartbeat / Alarm bei Ausfall eines Signals / Überschreitung von Grenzwerten
    Realisierbarkeit ohne zusätzliches technisches Knowhow machbar.
    Aufwand klein, maximal einen Tag.

Wichtigkeit niedrig, eher Convenience-Funktion.

- Statistische Auswertungen über alle Trips

  Realisierbarkeit alle Trips müssten auf dem mobilen Gerät aufgezeichnet werden. Ist einfacher auf der Server Seite zu realisieren, wäre aber grundsätzlich auf dem mobilen Gerät auch möglich.

  Aufwand gross, mindestens eine Woche

  Wichtigkeit gross, wenn JTourLive für sportliche Trainings eingesetzt werden soll.

- Akustische Ausgabe der Daten (Text to Speech)

  Realisierbarkeit sehr einfach, wenn ein Speech API vorhanden ist, ansonsten eine grosse Herausforderung.

  Aufwand gross, mindestens eine Woche

  Wichtigkeit gering, aber vielleicht ein gutes Verkaufsargument.

- Picture Producer: EXIF-Daten mit Zeitstempel und Position einfügen

  Realisierbarkeit einfach realisierbar, sofern das Mobiltelefon die Bilder als JPEG speichert

  Aufwand klein, ca. 1/2 Tag

  Wichtigkeit relevant, Bilder können so auf einer gefahrenen Strecke einem genauen Ort auf der Karte zugewiesen werden.

- Vom User konfigurierbare Views (z.B. XY-Diagram mit RPM, Gang und Speed)

  Realisierbarkeit durch eine erweiterte Menüstruktur möglich.

  Aufwand gross, ca. eine Woche

  Wichtigkeit gross, für die Akzeptanz im Markt, da jeder User die Software für seinen individuellen Einsatz anpassen kann.

- Weitere Producer

  - Video Stream

    Realisierbarkeit der Kern wurde so konstruiert, dass neben Datenpunkten auch Streams unterstützt werden. Dieser Support muss in die Sinks eingebaut und vervollständigt werden.

    Aufwand gross, ca. 1 Woche

    Wichtigkeit relevant, wenn die Wünsche der Fahrlehrer abgedeckt werden sollen.

  - Acceleration Sensor

    Realisierbarkeit auf ganz neuen Geräten welche das MIDP 2.1 API mit dem Paket *javax.microedition.sensor* des [24]JSR 256 unterstützen.

    Aufwand gering, 1-2 Tage

    Wichtigkeit relevant für die Ausbildung von VIP-Fahrern, wo die Beschleunigung während einer Fahrt einen bestimmten Grenzwert nicht übersteigen darf.

  - Audioaufnahme

    Realisierbarkeit auf praktisch allen mobilen Geräten umsetzbar. Evtl. ist mit einer Performance Einbusse zu rechnen.

    Aufwand gross, ca. 1 Woche

    Wichtigkeit relevant, wenn z.B. Kommentare des Fahrlehrers während einer Fahrt aufgezeichnet werden sollen.

- Persönliches Training
  - Trainings-Assistent

    Realisierbarkeit ohne spezielle technische Voraussetzungen möglich
    Aufwand sehr gross, ca. 2 Wochen
    Wichtigkeit gross, wenn JTourLive im Sportbereich eingesetzt werden soll.

- Fuhrparkverwaltung
  - Erfassung ob Firmen- oder Privatfahrt

    Realisierbarkeit ohne spezielle technische Voraussetzungen möglich.
    Aufwand gross, mindestens 2 Wochen, da vorallem die Server Seite massgebend erweitert werden muss.
    Wichtigkeit relevant, wenn JTourLive für Flottenmanagement usw. eingesetzt werden soll.
  - Download von Einstellungen / Remote Configuration

    Realisierbarkeit ohne spezielle technische Voraussetzungen möglich.
    Aufwand gering, ca 1/2 Tag
    Wichtigkeit gering, da das Runterladen einer Konfiguration nur bei einer Erstinstallation benötigt wird.

A Benutzeranleitung

A.1 Erste Schritte

A.1.1 Installation

Um die JavaME TourLive Anwendung auf einem mobilen Gerät zu installieren, muss die TourLive.jar Datei auf das mobile Gerät übermittelt werden. Diese Datei kann z.B. per Bluetooth an das mobile Gerät gesandt werden oder sie kann direkt vom mobilen Gerät aus dem Internet herunter geladen werden.

Um den Installationsvorgang zu starten, muss lediglich die TourLive.jar Datei angewählt werden. Während der Installation erscheinen mehrere Sicherheitsabfragen, welche alle mit 'ja', 'Fortfahr.', 'Wählen' oder 'OK' bestätigt werden können. Speziell wenn bereits eine ältere Version auf dem mobilen Gerät installiert war, sollten die Einstellungen gespeichert und beibehalten werden, da sonst alle zuvor gemachten Einstellungen verloren gehen.

A.1.2 Basiseingaben

Nach der Installation der JTourLive Anwendung sollten mindestens folgende Einstellungen gemacht werden:

1. *Sprache:* Unter Einstellungen>Sprache der Anwendung kann die Sprache auf Englisch oder auf Deutsch eingestellt werden.
2. *Angaben zur Person:* Unter Einstellungen>Puls sollte das *Geschlecht*, das *Alter*, das *Körpergewicht* und die *Körpergrösse* eingegeben werden, damit das Puls-Modul den Pulsbereich und den Kalorienverbrauch korrekt berechnen kann.
3. *Masseinheiten:* Unter Einstellungen>Anzeige>Masseinheiten kann die Darstellung zwischen dem *metrischen* und dem *imperialen* System umgestellt werden.

A.2 Vor der Aufzeichnung

Vor einer Aufzeichnung sollten folgende Einstellungen vorgenommen werden:

A.2.1 Aufnahmeintervall

Unter Einstellungen>Allgemein kann das *Zeitintervall* eingestellt werden, in welchem die Daten von den verschiedenen Datenquellen auf dem Bildschirm aktualisiert und auf dem mobilen Gerät aufgezeichnet oder zum Server übermittelt werden sollen.

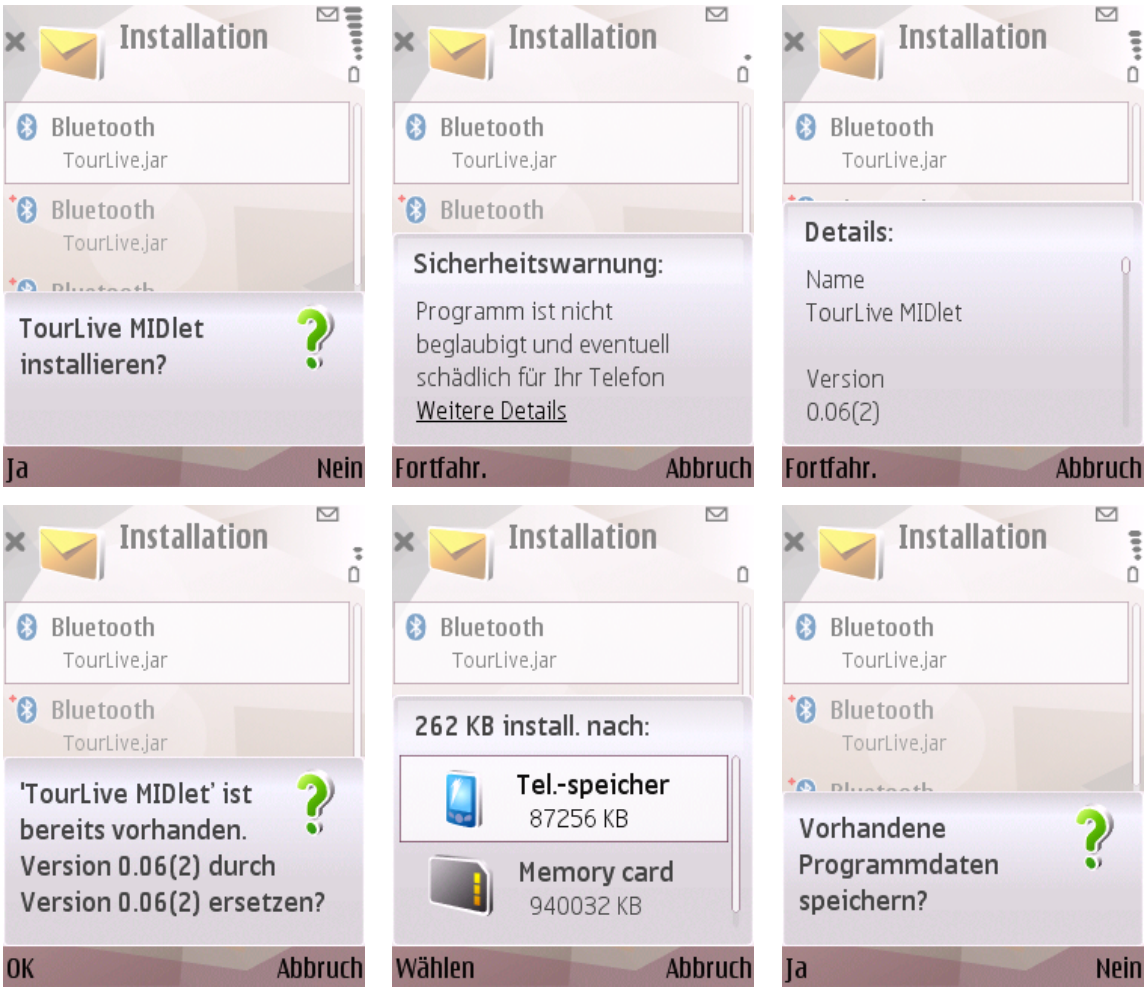

Abbildung A.1: Installation von JTourLive

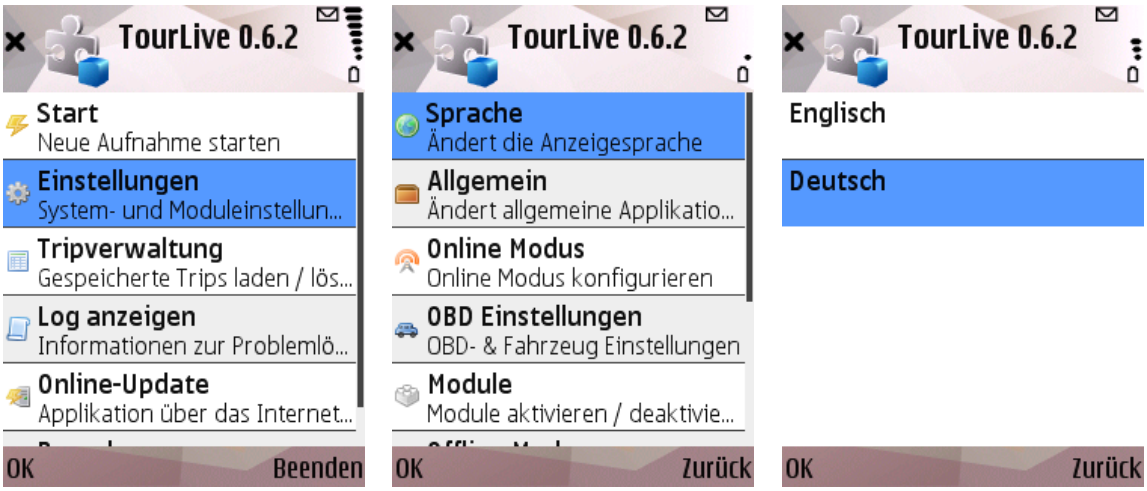

Abbildung A.2: Einstellung der Sprache

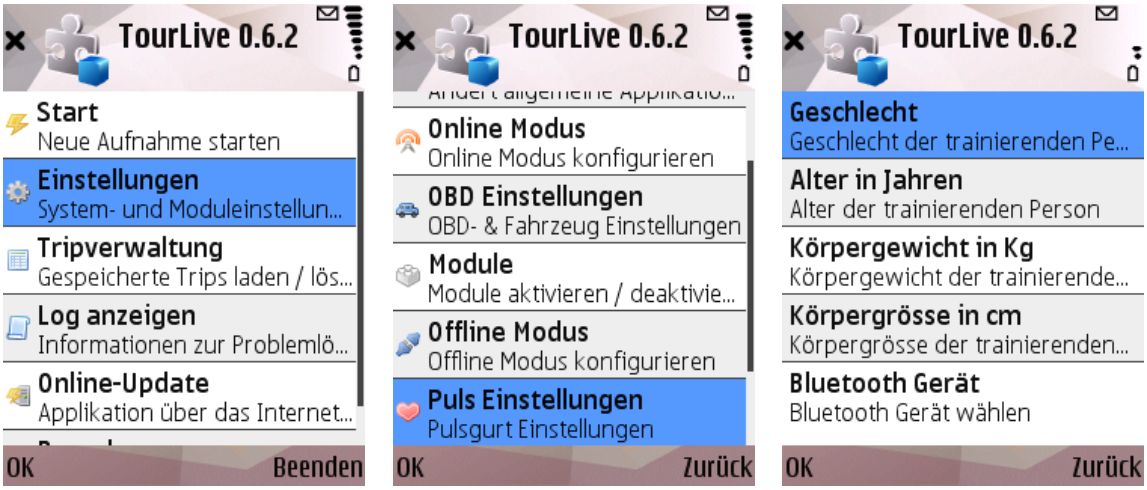

Abbildung A.3: Angaben zur Person

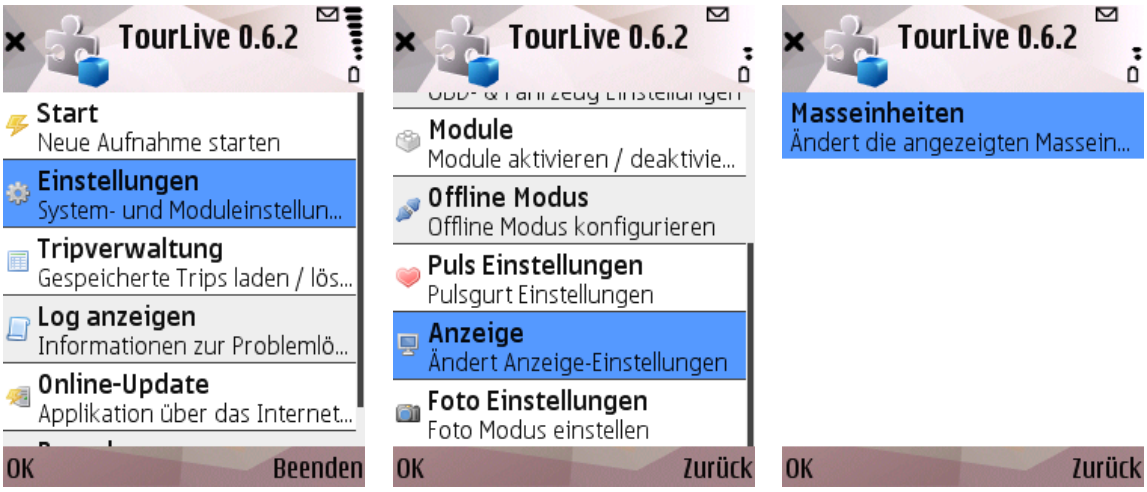

Abbildung A.4: Einstellung der Masseinheiten

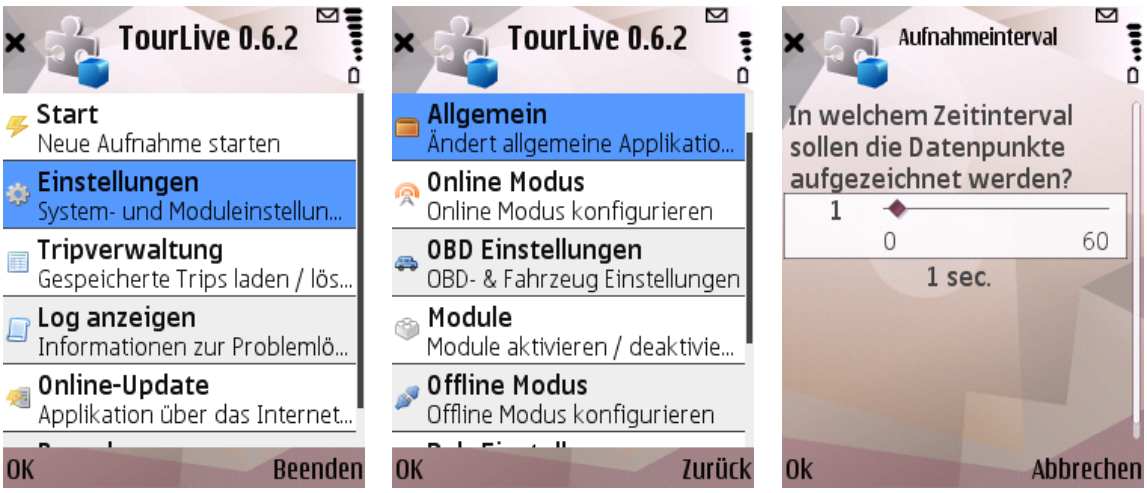

Abbildung A.5: Einstellung des Aufnahmeintervalls

A.2.2 Module Ein-/Ausschalten

Vor einer Aufzeichnung sollten die gewünschten Module ein- und die nicht benötigten Module ausgeschaltet werden.

A.2.2.1 GPS

Das GPS Modul wird benötigt, um die GPS-Position, die GPS-Höhe, die GPS-Geschwindigkeit und die GPS-Richtung aufzuzeichnen.

Unter Einstellungen>Module>GPS kann das GPS Modul mit *Ja* ein- oder mit *Nein* ausgeschaltet werden.

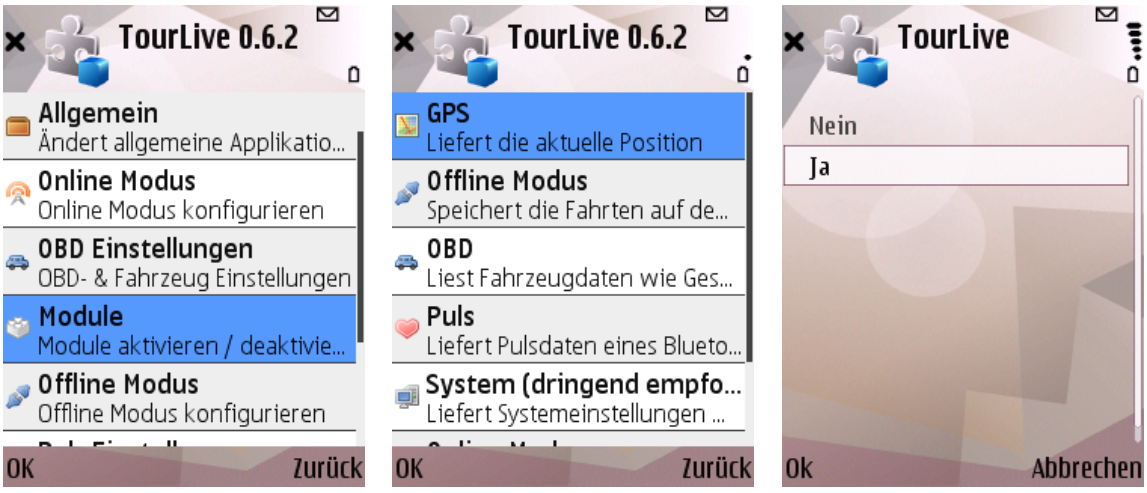

Abbildung A.6: Ein-/Ausschalten des GPS Moduls

A.2.2.2 Offline Modus

Das Offline Modus Modul wird verwendet, um die aufgezeichneten Daten in einer Datei abzuspeichern, wenn z.B. keine synchrone Übermittlung zum Server über das Internet möglich ist. Diese abgespeicherten Dateien können zu einem späteren Zeitpunkt z.B. über WLAN an den Server übermittelt werden.

Unter Einstellungen>Module>Offline Modus kann das Offline Modus Modul mit *Ja* ein- oder mit *Nein* ausgeschaltet werden.

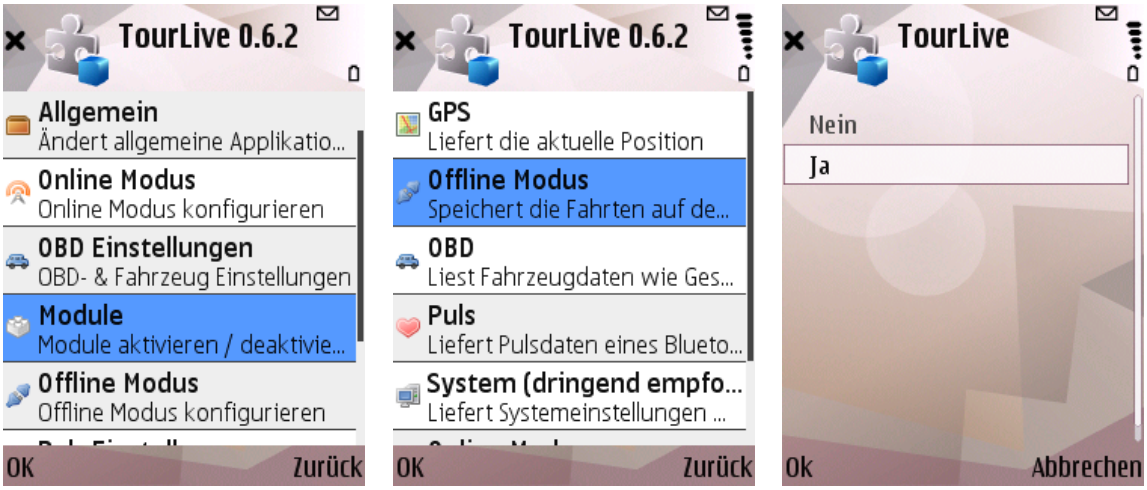

Abbildung A.7: Ein-/Ausschalten des Offline Moduls

A.2.2.3 OBD

Das OBD-Modul wird verwendet, um Daten aus einer OBD-II-Schnittstelle eines fahrenden Fahrzeuges auszulesen. Zu den OBD-Daten gehören die Geschwindigkeit, die Drehzahl, die Distanz, der Gang, der Verbauch und die Schubabschaltung.

Unter Einstellungen>Module>OBD kann das OBD-Modul mit *Ja* ein- oder mit *Nein* ausgeschaltet werden.

A.2.2.4 Puls

Das Puls Modul wird verwendet, um die vom Bluetooth Pulsgurt *Spurty-Brustgurt* der Firma mobimotion GmbH [20] übermittelte Herzfrequenz aufzuzeichnen.

Unter Einstellungen>Module>Puls kann das Puls Modul mit *Ja* ein- oder mit *Nein* ausgeschaltet werden.

A.2.2.5 System

Das System Modul stellt eine Übersicht über alle verwendeten Module und deren Status dar. Diese Übersicht ist sehr nützlich um zu sehen, welche Geräte erfolgreich mit dem

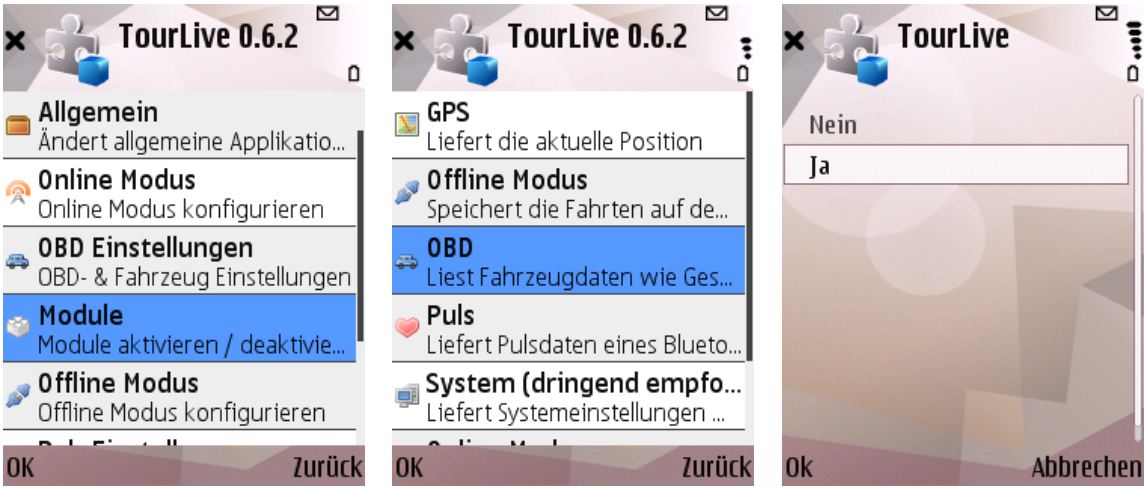

Abbildung A.8: Ein-/Ausschalten des OBD-Moduls

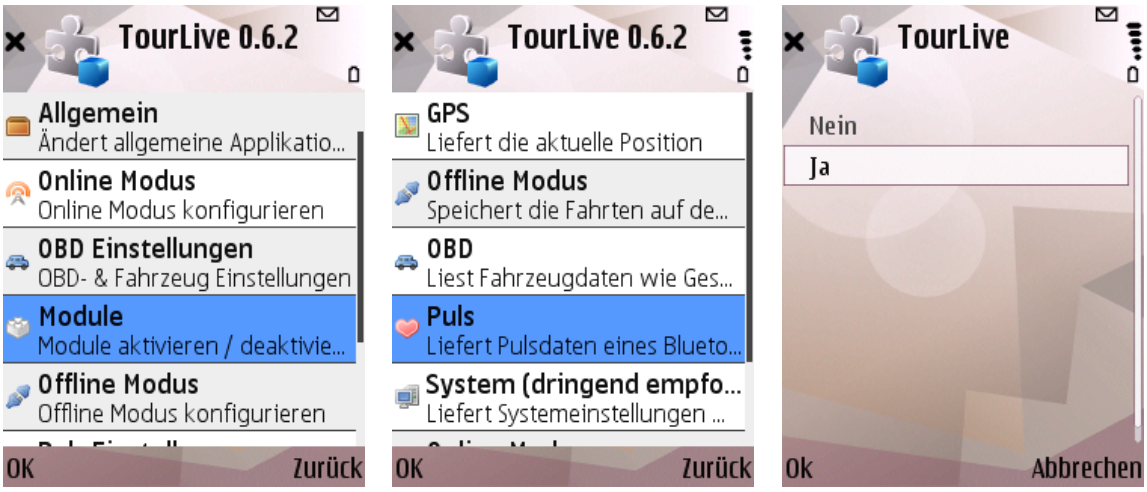

Abbildung A.9: Ein-/Ausschalten des Puls Moduls

aufzeichnenden mobilen Gerät verbunden sind.

Unter Einstellungen>Module>System kann das System Modul mit *Ja* ein- oder mit *Nein* ausgeschaltet werden.

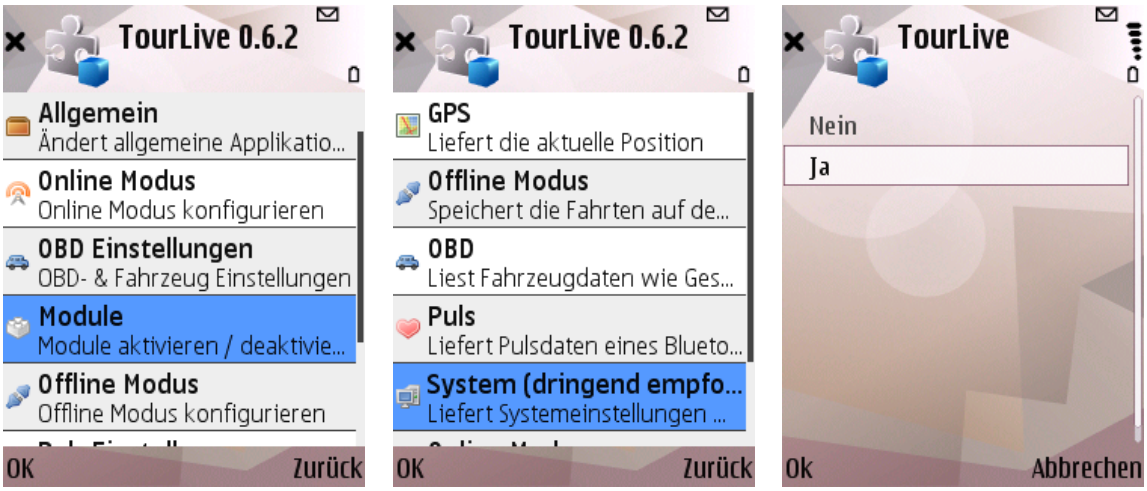

Abbildung A.10: Ein-/Ausschalten des System Moduls

A.2.2.6 Online Modus

Das Online Modus Modul wird verwendet, wenn die Daten direkt während der Aufzeichnung synchron an einen Server übermittelt werden sollen.

Unter Einstellungen>Module>Online Modus kann das Online Modus Modul mit *Ja* ein- oder mit *Nein* ausgeschaltet werden.

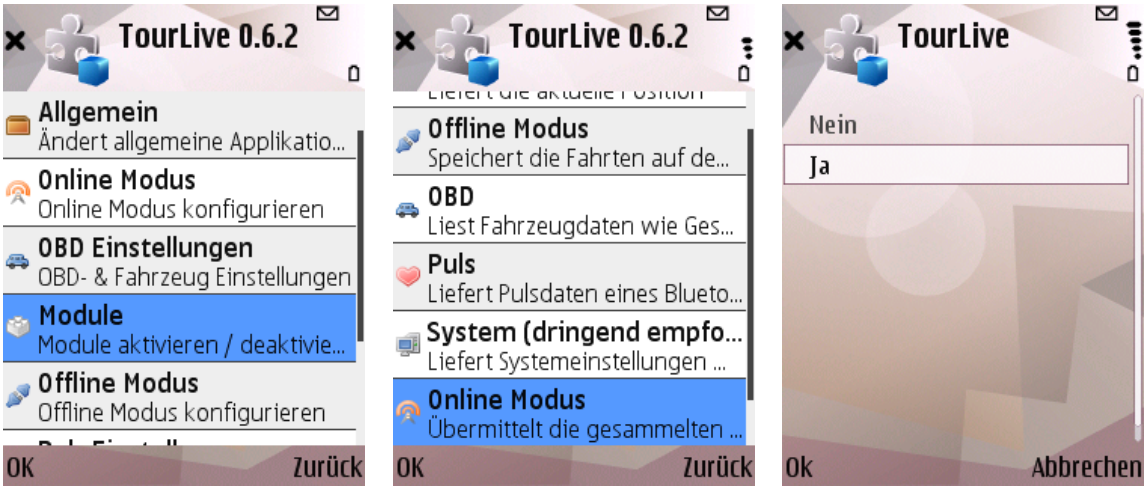

Abbildung A.11: Ein-/Ausschalten des Online Moduls

A.2.2.7 Foto

Das Foto Modul wird verwendet, um während der Aufzeichnung Bilder über die im mobilen Gerät integrierte Kamera zu schiessen.

Unter Einstellungen>Module>Foto kann das Foto Modul mit *Ja* ein- oder mit *Nein* ausgeschaltet werden.

Abbildung A.12: Ein-/Ausschalten des Foto Moduls

A.2.3 Geräte verbinden

Wenn während der Aufzeichnung externe Geräte zum Einsatz kommen sollen, müssen diese vorher ausgewählt werden.

A.2.3.1 OBD

Wenn das OBD-Modul eingeschaltet ist, muss vor der ersten Aufzeichnung oder wenn ein neuer OBD-II-Bluetooth-Adapter zum Einsatz kommen soll vorher der zu verwendende OBD-II-Bluetooth-Adapter ausgewählt werden. Dies geschieht unter Einstellungen>OBD Einstellungen>Bluetooth Gerät. Das mobile Gerät enumeriert alle erreichbaren Bluetooth-Geräte. Damit der OBD-II-Bluetooth-Adapter gefunden wird, muss er also bereits eingesteckt und mit Strom versorgt sein. Nach dem Auswählen wird oft nach einem Passwort verlangt. Dies ist meistens ein Standard Passwort welches 0000, 1111 oder 1234 lautet.

A.2.3.2 Puls

Wenn das Puls-Modul eingeschaltet ist, muss vor der ersten Aufzeichnung oder wenn ein anderer Bluetooth-Pulsgurt zum Einsatz kommen soll vorher der zu verwendende Bluetooth-Pulsgurt ausgewählt werden. Dies geschieht unter Einstellungen > Puls Einstellungen > Bluetooth Gerät. Das mobile Gerät enumeriert alle erreichbaren Bluetooth-Geräte. Damit

Abbildung A.13: Konfigurieren des OBD-II-Bluetooth-Adapters

der Bluetooth-Pulsgurt gefunden wird, muss er also geladen und eingeschaltet sein. Nach dem Auswählen wird oft nach einem Passwort verlangt. Dies ist meistens ein Standardpasswort welches 0000 oder 1234 lautet.

Abbildung A.14: Konfigurieren des Bluetooth-Pulsgurtes

A.2.4 Server konfigurieren

Wenn das Online Modus Modul eingeschaltet ist und während einer Aufzeichnung die Daten direkt an einen Server übermittelt werden sollen, muss vorher die *Server-URL* und der *Mobile-Schlüssel* eingegeben werden. Unter Einstellungen>Online Modus>Server URL kann die *Server-URL* und unter Einstellungen>Online Modus>Mobiler Schlüssel der *Mobile-Schlüssel* eingegeben werden.

Abbildung A.15: Eingeben der Server Konfiguration

A.2.5 Speicherort wählen

Wenn das Offline Modus Modul eingeschaltet ist und die Daten während der Aufzeichnung für eine spätere Auswertung gespeichert werden sollen, muss noch der Speicherort festgelegt werden, wo die Aufzeichnung als Datei abgespeichert werden soll. Der Speicherort kann unter Einstellungen>Offline Modus>Speicherort angegeben werden.

Abbildung A.16: Konfigurieren des Speicherorts für den Offline Modus

A.2.6 Foto Modus einstellen

Wenn das Foto Modul eingeschaltet ist, sollte vor einer Aufzeichnung noch der Foto Modus auf *manuell* oder auf *automatisch* eingestellt werden. *Manuell* bedeutet, dass Fotos manuell während der Aufzeichnung ausgelöst werden können. *Automatisch* bedeutet, dass in regelmässigen Abständen automatisch Fotos gemacht und mit den anderen Daten aufgezeichnet

werden.

Abbildung A.17: Einstellen des Foto Mous

A.3 Während der Aufzeichnung

Folgende Operationen können während einer Aufzeichnung vorgenommen werden:

A.3.1 Ansichten wechseln

Während einer laufenden Aufzeichnung kann mit den Pfeiltasten nach links und nach rechts zwischen den verschiedenen Tabs gewechselt werden.

A.3.2 Benutzerbefehle

Benutzerbefehle sind Befehle, welche während einer laufenden Aufzeichnung ausgeführt werden können.

A.3.2.1 Gänge zurücksetzen

Wenn vor dem Aufzeichnungsstart das OBD-Modul eingeschaltet wurde, kann während der Fahrt mit *Gänge zurücksetzen* die bereits erkannten Gänge zurückgesetzt werden, wenn z.B. ein Gang falsch oder der Rückwärtsgang erkannt wurde.

A.3.2.2 Foto auslösen

Wenn vor dem Aufzeichnungsstart das Foto Modul eingeschaltet und der Foto Modus auf *manuell* eingestellt wurde, kann während der Aufzeichnung mit *Foto auslösen* ein Foto geschossen werden. Dabei muss beachtet werden, dass bei manchen mobilen Geräten vorher die Kameralinse geöffnet werden muss.

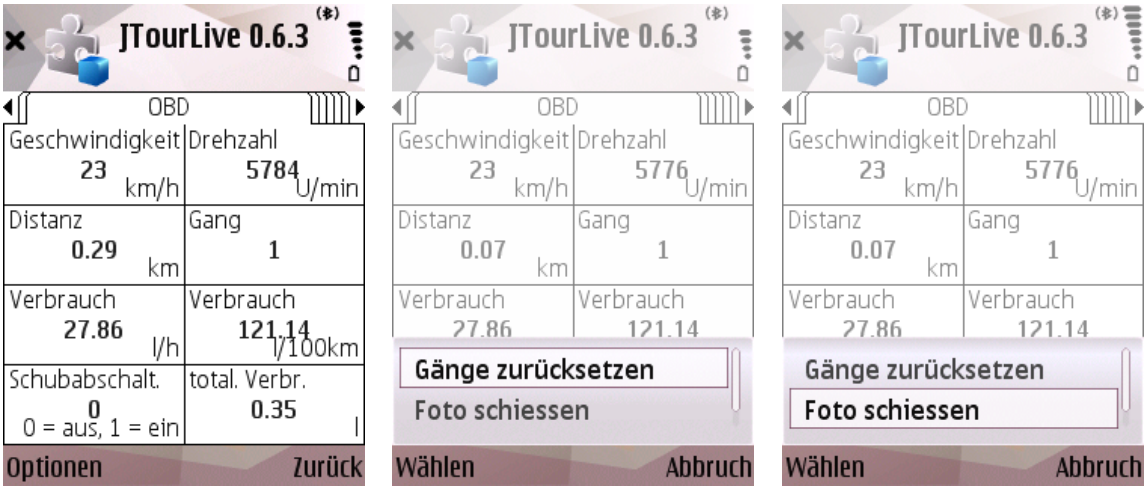

Abbildung A.18: Benutzerbefehle während der Aufzeichnung

A.3.3 Aufzeichnung beenden

Am Ende einer Fahrt kann die Aufzeichnung mit *zurück* beendet werden.

A.4 Nach der Aufzeichnung

Nach einer Aufzeichnung können alle Einstellungen wieder verändert, die Aufzeichnung übermittelt, wieder eine neue Aufzeichnung gestartet oder die Anwendung beendet werden.

A.4.1 Aufzeichnung übermitteln

Wenn während einer Aufzeichnung das Offline Modus Modul eingeschaltet war, wurde die Aufzeichnung in einer Datei abgespeichert. Diese kann im Nachhinein z.B. über ein WLAN über das Internet an den Server übermittelt werden. Dies geschieht unter Tripverwaltung. Dort kann die zu übermittelnde Datei ausgewählt und mit *Trip übermitteln* versendet werden.

A.5 Weitere Funktionen

Dies sind Funktionen, die sporadisch oder bei Fehlverhalten der Anwendung verwendet werden können

A.5.1 Log anzeigen

Wenn die Anwendung während einer Aufzeichnung ein Fehlverhalten gezeigt hat, gibt vielleicht die Logdatei Aufschluss über den Fehler. Die Logdatei kann unter *Log anzeigen* eingesehen werden. Bei Bedarf kann die Logdatei mit *an den Server übermitteln* für die Entwickler an den Server gesandt werden.

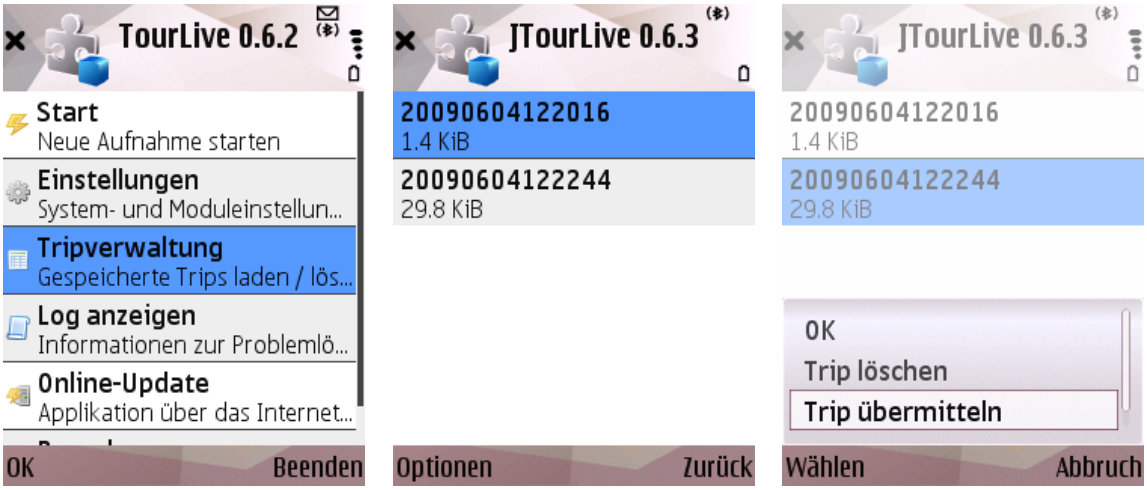

Abbildung A.19: Übermitteln von aufgezeichneten Fahrten

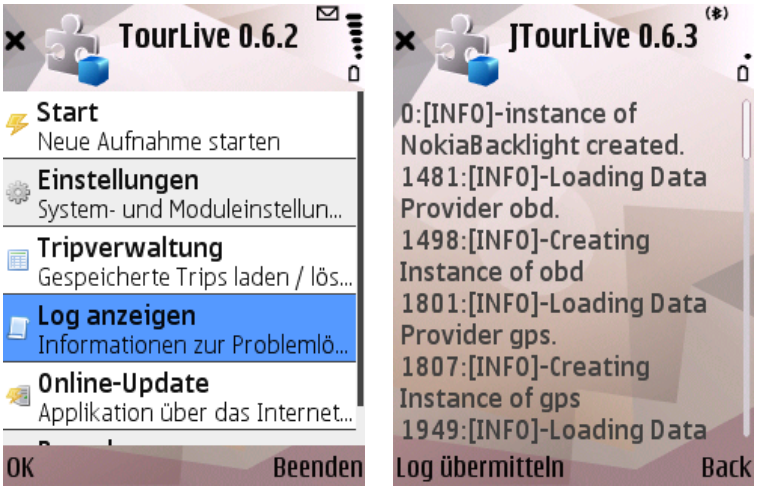

Abbildung A.20: Log Datei anzeigen und an den Server übermnitteln

A.5.2 Online Update

Die JTourLive Anwendung verfügt über eine Online Update Funktion. Unter *Online-Uptdate* kann überprüft werden, ob eine neue Version der JTourLive Anwendung zur Verfügung steht. Wenn dem so ist, kann sie mit *Ok* direkt installiert werden. Das *Online-Update* sucht auf dem unter den Online-Einstellungen angegebenen Server nach Aktualisierungen.

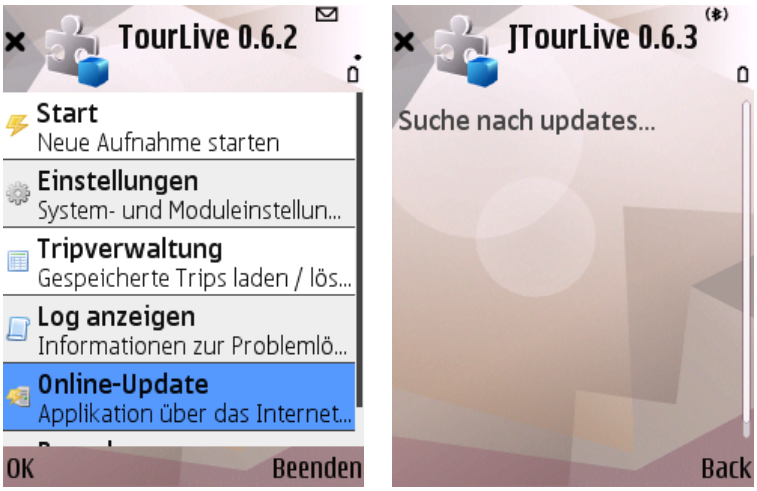

Abbildung A.21: JTourLive über das Online Update aktualisieren

B Administratoranleitung

B.1 Installation der Server-Komponente

Die Server-Komponente benötigt einen lauffähigen WebServer mit installiertem PHP und einer MySQL-Datenbank.

1. Alle Dateien aus src/server in ein nach aussen zugängiges Verzeichnis kopieren
2. config.default.php nach config.php kopieren und die Einstellungen anpassen:
   a) Defines für IMG_PATH und LOG_PATH anpassen
   b) Neues Secret generieren und im define SECRET_KEY austauschen
   ACHTUNG: Standard-Secret sollte nicht weiter verwendet werden, da sonst die Authentifizierung ausgehebelt werden kann
   c) $db_config anpassen
   d) $replication_hosts anpassen
3. Die unter IMG_PATH und LOG_PATH angegebenen Verzeichnisse müssen existieren und der Benutzer unter welchem der Web Server läuft muss schreibrechte darauf haben
4. Das Verzeichnis LOG_PATH sollte gegen den Zugriff von aussen abgesichert werden, falls dieses unterhalb des Web Roots ist
5. In der Konfiguration des Online-Modus die URL zum Server inkl. abschliessendem / eingeben (Bsp: http://localhost/jtourlive/)

C Entwickleranleitung

C.1 IDE einrichten

C.1.1 Eclipse (unter Linux)

Im folgenden Abschnitt wird beschrieben, wie die Eclipse IDE für die Entwicklung von JavaME Anwendungen installiert wird.

Voraussetzung Auf dem Entwicklungsrechner muss das Java Runtime Environment (JRE) Version 1.5.0 oder höher installiert sein. Für das Erzeugen und das Ausliefern der JAR Datei muss ant und Ruby installiert sein.

1. Download von Eclipse für Java Entwickler
   Unter http://www.eclipse.org/downloads/ kann die neuste Version für die entsprechende Plattform heruntergeladen werden.

2. Eclipse entpacken
   Die heruntergeladene Datei im gewünschten Ordner entpacken.

3. Download Wireless Tool Kit
   Ein Wireless Tool Kit herunterladen z.B. unter http://java.sun.com/products/sjwtoolkit/download.ht
   Es können aber auch beliebige andere verwendet werden.

4. Wireless Tool Kit (WTK) installieren
   Dem heruntergeladenen File die Berechtigung zur Ausführung erteilen und ausführen.
   Das Installationsskript verlangt einige Pfadangaben.

   ```
   > chmod +x sun_java_wireless_toolkit-2.5.2_01-linuxi486.bin.
     sh
   > ./sun_java_wireless_toolkit-2.5.2_01-linuxi486.bin.sh
   ```

5. Installation von Eclipse Device Software Development Platform (DSDP) Mobile Tools for Java (MTJ)
   Eclipse starten und unter *Help* > *Software Updates* die Update Site von http://download.eclipse.org/d
   mit *Add Site* hinzufügen. Anschliessend unter *Available Software* Mobile Tools for Java anwählen und installieren. Nach der Installation muss Eclipse neu gestartet werden.

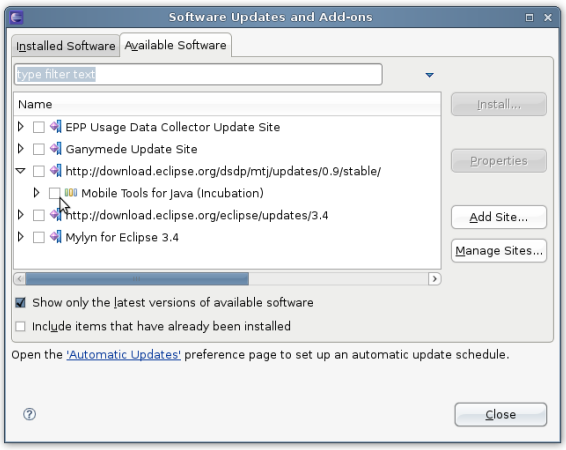

Abbildung C.1: MTJ Installations Dialog

6. Einbinden des WTK
   Am einfachsten erstellt man im Eclipse ein neues MIDlet Projekt und klickt bei *Configurations* auf *Add.*

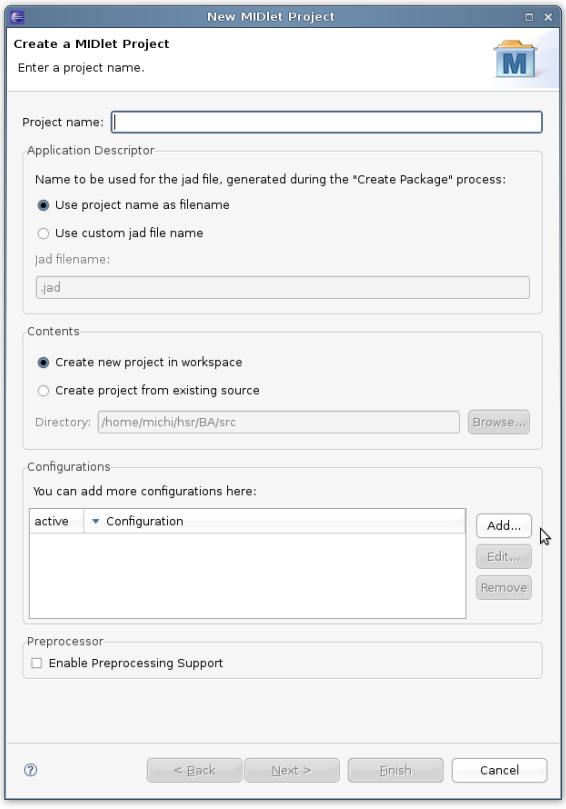

Abbildung C.2: MIDlet Projekt Dialog

Im darauf folgenden Dialog wählt man *Manage Devices* um das SDK und das Gerät

zu wählen.

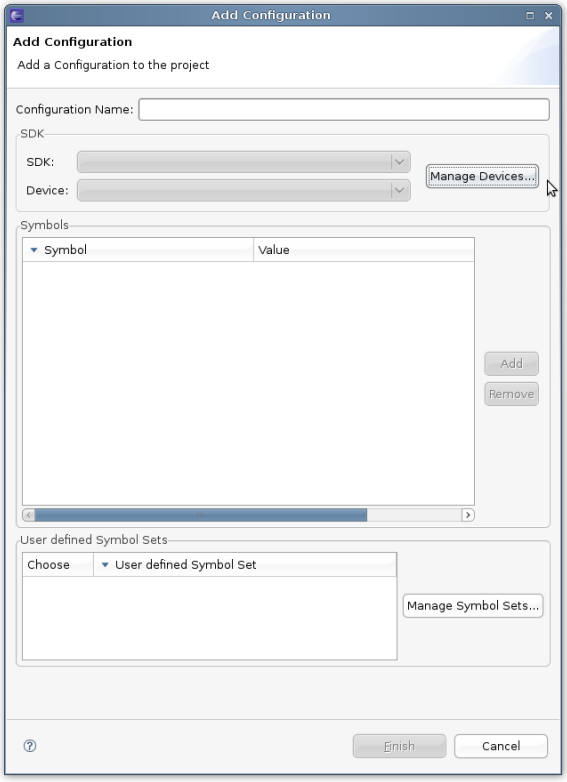

Abbildung C.3: Konfigurations Dialog

Im darauf folgenden Dialog kann man mit *Import* den Pfad des WTKs angeben. Eclipse durchsucht den Ordner nach vorhandenen Geräten und zeigt diese an. Anschliessend kann dann aus der Liste das gewünschte Gerät mittels Checkbox als Default markiert werden.

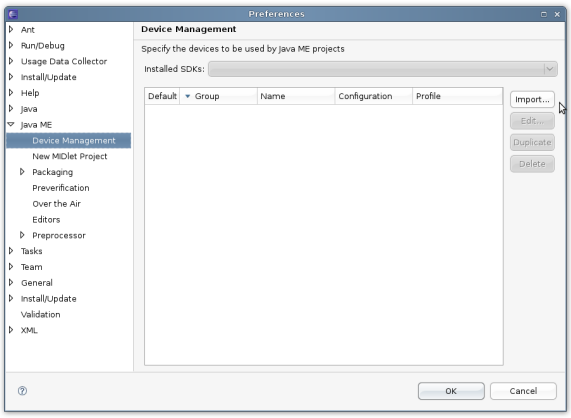

Abbildung C.4: WTK Import Dialog

Eclipse zeigt im Import Dialog alle Geräte an, die es unter dem WTK Pfad gefunden hat. Am besten importiert man gleich alle zur Verfügung stehenden Geräte, damit man später zwischen diesen wechseln kann.

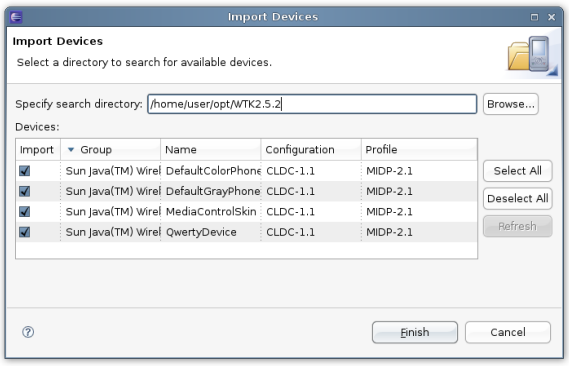

Abbildung C.5: Importierte Geräte

Die Installation wird mit *Finish* beendet

7. Download von Antenna
   Unter http://sourceforge.net/project/showfiles.php?group_id=67420 kann die aktuelle Version von Antenna heruntergeladen werden.
8. Installation von Antenna
   Für die Ant Buildfiles wird für MIDlets die Ant Erweiterung Antenna verwendet. Für den Buildfile Export muss im Eclipse zuerst noch unter *Window* > *Preferences* > *Java ME* der Pfad zum Antenna JAR und das Root Verzeichnis des WTKs angegeben werden. Anschliessend können die Build Files mit einem Rechtsklick auf das Projekt > *Mobile Tools for Java* > *Export Antenna Build Files* exportiert werden.

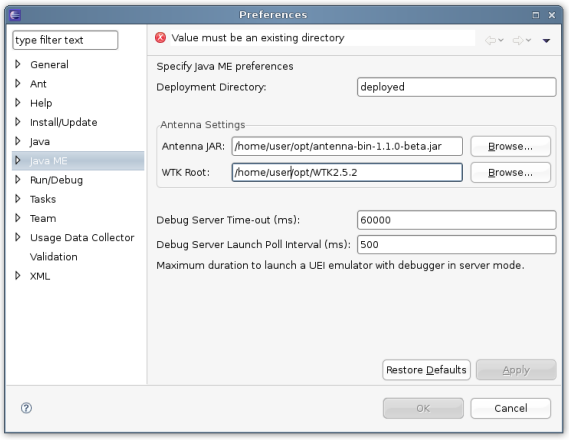

Abbildung C.6: Installation von Antenna

9. Importieren der Projekte

Über File -> Import -> Existing Projects into Workspace können die bestehenden Projekte in Eclipse importiert werden. Dazu sollte Eclipse im richtigen Workspace (TourLive/src) gestartet und die Option *Copy projects into workspace* nicht angewählt sein.

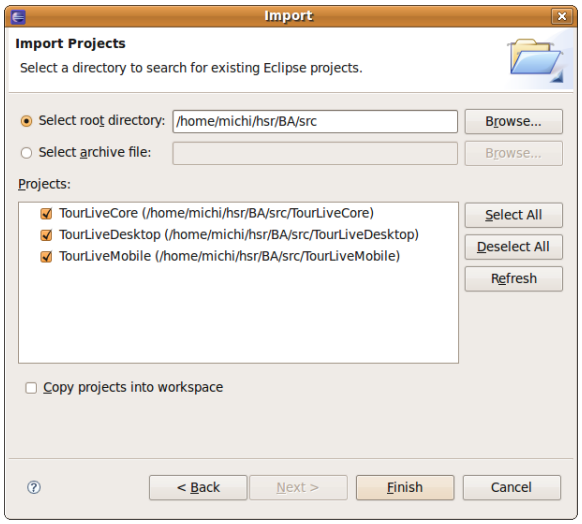

Abbildung C.7: Importieren der JTourLive Projekte

10. Einrichten des JDKs
    Da das Projekt TourLiveCore nur die Klassen des CLDC Version 1.1 verwenden darf, muss für die Entwicklung das JDK speziell eingerichtet werden. Dazu öffnet man in den *Project Properties* die Seite *Java Build Path*, und wählt unter *Libraries* die JRE *JavaME* aus.

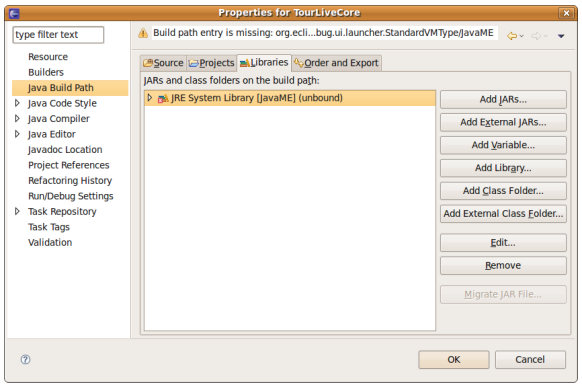

Abbildung C.8: Einrichten des Build Pfades

Nach einem Klick auf *Edit* und im folgenden Dialog auf *Installed JREs* erscheint der folgende Dialog, bei dem man mit *Add* ein neues JRE eintragen kann:

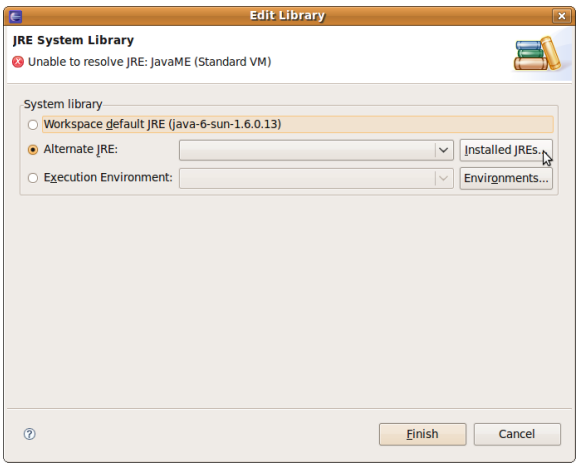

Abbildung C.9: Installierte JREs

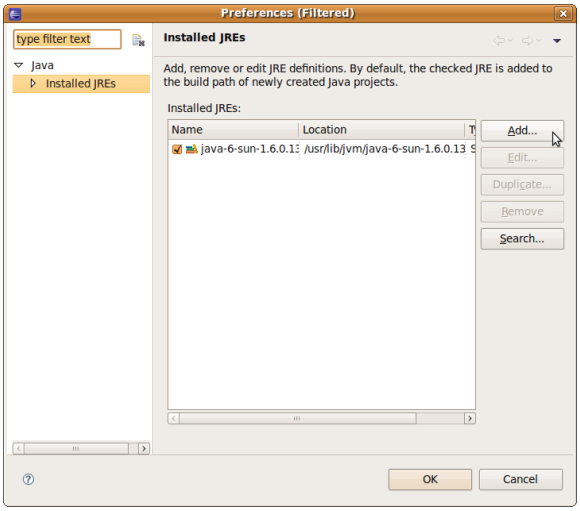

Abbildung C.10: Hinzufügen einer JRE

Im folgenden Wizard wählt man *Standard VM* und gibt im folgenden Schritt unter *JRE home* den Pfad eines 32bit JREs an. Der *JRE name* ist JavaME und die *Default VM arguments* können leer gelassen werden.

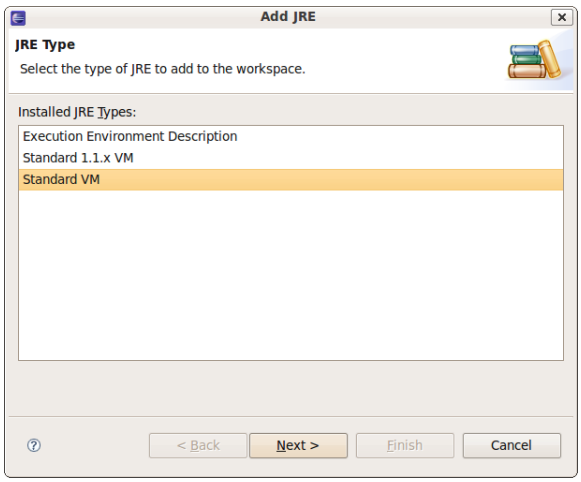

Abbildung C.11: Installierte JREs

Unter den System Libraries müssen alle bestehenden Einträge gelöscht werden. Danach fügt man über *Add external JARs* alle Bibliotheken aus dem Verzeichnis *WTK_ROOT/lib* hinzu und löscht anschliessend die Libraries *cldcapi10.jar*, *midpapi10.jar* und *midpapi21.jar*.

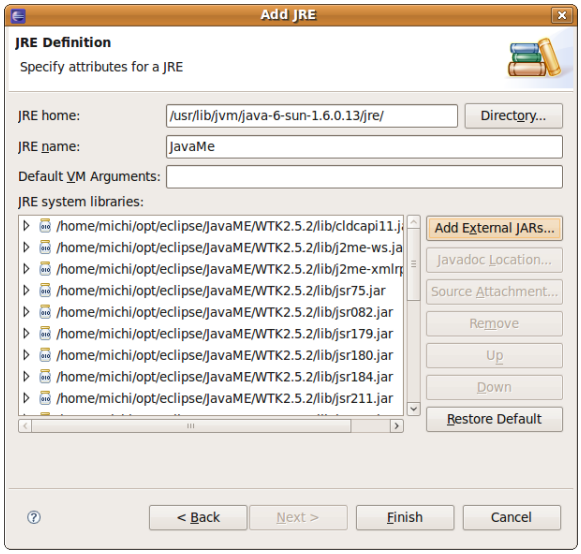

Abbildung C.12: JavaME JRE

Jetzt können die Fenster alle geschlossen werden und die Projekte TourLiveCore und TourLiveDesktop sollten fehlerfrei kompilieren. Falls das TourLiveDesktop Projekt Fehler anzeigt, muss vielleicht die JRE auf die “normale”, nicht JavaME JRE umgestellt werden, da möglicherweise die Workspace default JRE auf JavaME umgestellt wurde.

11. Beheben von Konfigurationsproblemen des TourLiveMobile-Projekts

Falls das TourLiveMobile-Projekt noch den Fehler *No device definition is associated to this project* anzeigt, ist ein weiterer Schritt nötig. Das Problem ist, dass bei unterschiedlichen Konfigurationen die *Device configuration* nicht identisch ist, weshalb Eclipse das Projekt nicht kompilieren und ausführen kann. Um das Problem zu beheben, öffnet man die Eigenschaften vom TourLiveMobile Projekt und öffnet die *Java ME*-Einstellungen. Hier klickt man unter *Configurations* auf *Add* und fügt eine neue Konfiguration hinzu, die in diesem Projekt noch nicht existiert (z.B. *DefaultGrayPhone*).

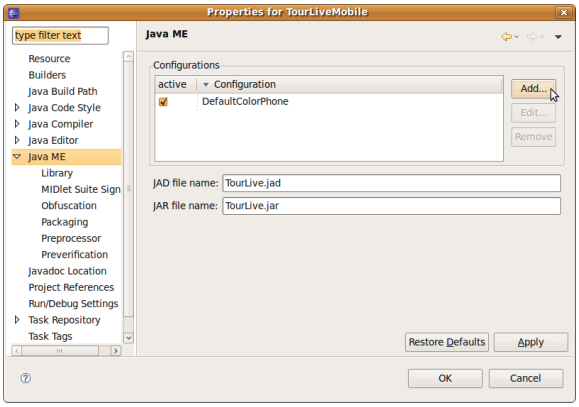

Abbildung C.13: Konfiguration hinzufügen

Danach kann man das *DefaultColorPhone* löschen und wie oben beschrieben erneut hinzufügen. Wenn die Konfiguration neu erstellt wurde, sollte man den Standard wieder auf das ColorPhone setzen und das GrayPhone wieder entfernen.

12. Damit das automatische Erzeugen und Ausliefern per ant-Script funktioniert muss im src/TourLiveMobile/ und im src/TourLiveCore/ eine Datei namens build.local.properties erstellt werden.
In beiden muss die Variable midp.home auf den Pfad zum aktuellen WTK gesetzt werden.

```
midp.home = /home/user/.../WTK2.5.2
```

Wenn das JAR mit *ant deploy* automatisch ausgeliefert werden soll, kann unter Linux im build.local.properties unter src/TourLiveMobile/ noch der deploy.cmd Variable das entsprechende Kommando zugewiesen werden.
z.B.:

```
deploy.cmd = /usr/bin/bluetooth-sendto dist/TourLive.jar --
    device 'MA:CA:DR:ES:SE:00'
```

Wenn das JAR zusätzlich noch in ein spezielles Verzeichnis ausgeliefert werden soll (z.B. auf einen Webserver) kann zusätzlich noch die Variable deploy.dir gesetzt werden.

```
deploy.dir = /var/www/tourlive/deploy/
```

C.1.2 NetBeans (unter Linux)

In folgendem Abschnitt wird beschrieben, wie die NetBeans IDE für die Entwicklung von JavaME MIDlets eingerichtet wird.

Voraussetzung Auf dem Entwicklungsrechner muss ein aktuelles (zur Zeit der Arbeit JDK 6.0) Java JDK vorinstalliert sein. Für den Midlet-Emulator von NetBeans muss zwingend eine 32-Bit Version des JDK installiert werden.
Bezugsquelle: http://developers.sun.com/downloads/

1. Download von NetBeans
   Unter http://www.netbeans.org/downloads/index.html kann NetBeans für die gewünschte Sprache und die gewünschte Plattform heruntergeladen werden. Für die Entwicklung von MIDlets empfiehlt sich die Variante "Java", bei der folgende Pakete bereits dabei sind:

   - Java SE

   - Java Web and EE

   - Java ME

   - GlassFish V2 UR2

   - GlassFish v3 Prelude

   - Apache Tomcat 6.0.18

   Alternativ kann auch die Variante "Java SE" gewählt und das Paket Java ME über den Plugin Manager von NetBeans nachinstalliert werden.

2. Setzen des executable Bit auf der Installationsdatei im Verzeichnis des Downloads.

   ```
   > chmod +x netbeans-6.5-ml-java-linux.sh
   ```

3. Ausführen der Installationsdatei im Verzeichnis des Downloads.

   ```
   > ./netbeans-6.5-ml-java-linux.sh
   ```

   Das Installationsskript startet ein Installations Wizard, über den alle nötigen Angaben zur Installation vorgenommen werden können.

Abbildung C.14: Willkommensseite des NetBeans Installationsprogramms

Wenn der Wizard das vorinstallierte Java JDK nicht automatisch findet, kann der Pfad zum JDK unter *Customize...* angegeben werden.
Weiter mit *next*.

4. Die Lizenzbestimmungen von Sun müssen akzeptiert werden.

Abbildung C.15: Lizenzvereinbarungen von NetBeans

5. Das Installationsverzeichnis kann gewählt werden

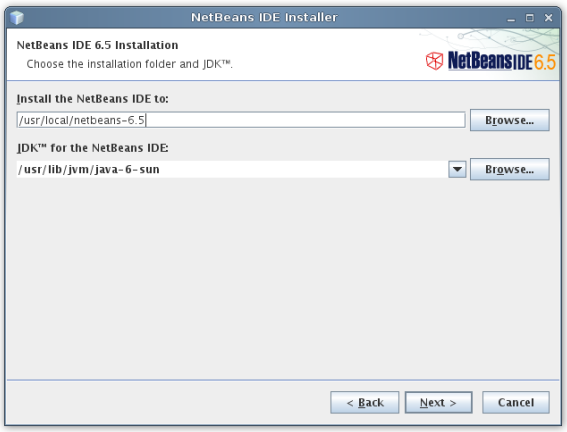

Abbildung C.16: Auswahl des Installatiosnverzeichnisses

6. Es folgen im Fall der Variante "Java" weitere Wizard Masken für GlassFish, welche für JavaME MIDlets nicht von Bedeutung sind und beim Default belassen werden können.
7. Die Installations-Zusammenfassung muss mit *install* bestätigt werden.

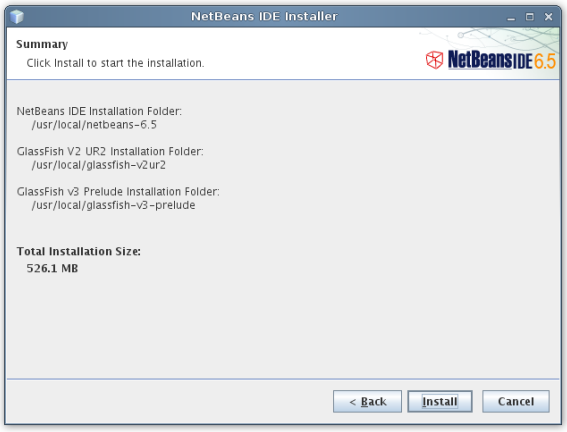

Abbildung C.17: Zusammenfassung der Installationsschritte

Der Installationsprozess beginnt.

8. Optional kann bei statistischen Erhebungen von NetBeans teilgenommen werden und NetBeans elektronisch registriert werden.
   Mit *Finish* wird die Installation beendet.
9. Wenn auf dem Entwicklungsrechner mit einer 64-Bit Version des JDK entwickelt wird und die 32-Bit Version lediglich für den MIDlet Emulator eingesetzt wird, muss in allen Binärdateien die *pathtowtk* Variable auf den Pfad der 32-Bit Version gesetzt werden.

```
/home/user/netbeans-6.5/mobility8/WTK2.5.2/bin> sed -e 's@^
    javapahttowtk=$@pathtowtk=/path/to/32-bit-jdk/bin/@' -i *
```

10. NetBeans ist nun fertig installiert und kann für die Entwicklung von MIDlets verwendet werden.

D Risiko-Analyse

Risiko Analyse

Projektname:	TourLive
Projektmanager:	Stephan Hauser, Michael Wagner
Datum Kalkulation:	12/03/09

Risiko Bewertungen

Risk ID	Risiko	Auswirkung	Max. Schaden in Stunden	Wahrschein-lichkeit des Eintreffens	Massnahme	Aufwand der Massnahmen in Stunden	Rest-Wahrscheinlichkeit des Eintreffens nach Massnahmen	Gewichteter Schaden in Stunden	Priorität
R01	Nichtbestehen der Bachelor-Arbeit	Verlängerung der Studienzeit um 1 Semester	370	5.00%	regelmässige Rückfragen beim betreuenden Dozenten, ob Zwischenstand o.k.	16	0.50%	2	sehr hoch
R02	schlechte Note für die Bachelor-Arbeit	Negativer Einfluss auf Notenschnitt des gesamten Studiums.	-	12.50%	Erfüllen der gestellten Aufgaben und umsetzen von einigen nicht zwingenden Wünschen Extras darüber hinaus.	60	3.00%	-	hoch
R03	JavaME API untersützt Zugriff auf bestimmte Resourcen oder HW nicht	Fearures können nicht implementiert werden oder müssen in nativem Code realisiert werden	-	80.00%	frühe Abklärungen der Möglichkeiten des JavaME API und evtl. Einschränkung des Funktionsumfanges	32	4.00%	-	hoch
R04	Datenübertragung auf Server nur mit Einschränkungen möglich	Keine Echtzeitübermittlung oder üerlaufende Buffer	32	30.00%	Frühe Abklärung der Übertragungsmöglich keiten mit Prototyp	16	3.00%	1	hoch
R05	Zugriff auf Resourcen oder HW ist Hersteller spezifisch	Es müssen verschiedene SW Varianten erstellt werden	40	70.00%	frühe Tests auf versch. Geräten	16	7.00%	3	mittel
R06	Schnittstelle verhalten sich nicht erwartungsgemäss	Unerwartetes Verhalten und inkorrekte Messwerte	80	50.00%	Früh in Schnittstellen Doku einlesen und Prototypen erstellen	40	5.00%	4	mittel
R07	Teammitglied fällt aus	Projekt kann nicht in geplanter Zeit umgesetzt werden	370	1.00%	-	-	1.00%	4	niedrig
R08	Zielplattformen nicht genug leistungsfähig	Software läuf langsam und kann Messintervalle nicht einhalten	80	20.00%	Software für knappe Resourcen auslegen	16	5.00%	4	niedrig
R09	Ausfall von Entwicklungs-Systemen	Verlust von Sourcecode oder Dokumentation	720	0.50%	verteiltes Backup mittels SVN	8	0.05%	0	niedrig
	Total Kosten in Arbeitspaketen enthalten					128			
	Total Rückstellungen							16	

Abbildung D.1: Risiko Management

E Persönliche Erfahrungen

E.1 Stephan Hauser

Die Entwicklung einer Software-Lösung auf Basis von JavaME ist eine Entscheidung, die nicht leichtfertig getroffen werden sollte. Auf der einen Seite stehen viele Geräte, die JavaME unterstützen und damit ein breites Publikum für die Applikation. Auf der anderen Seite jedoch wiegen unterschiedliche Deutungen der Standards, limitiertes und je nach Gerät und Hersteller unterschiedliches Feature-Set, und eine mühsame Entwicklung sehr schwer. So ist es z.B. nur möglich mit Java 1.4 zu arbeiten, was bedeutet, dass auf Generics verzichtet werden muss. Des weiteren steht mit MIDP 2.0 nur ein kleines Subset der Library zur Verfügung, welches auch wichtige Dinge wie z.B. *ArrayList*, oder *String.format* vermissen lässt.

Die explizite Verwendung eines Software-Entwicklungsmodells wie RUP ist zwar nicht erfolgt, jedoch haben wir uns bei der Planung und Durchführung im Groben an die Phasen gehalten. Vor allem der Inception ist dabei ein grosser Teil der Entwicklung zuzuschreiben, bei der wir mit verschiedenen Prototypen sehr schön die Machbarkeit der einzelnen Teile zeigen konnten. Diese Vorgehensweise hat uns vor allem auch bei der Zeitplanung sehr stark geholfen, wodurch es überhaupt möglich wurde ein so grosses Projekt erfolgreich und ohne Zeitprobleme abzuschliessen.

Auch aus dieser Arbeit konnte ich wieder einige Erfahrungen gewinnen. So habe ich erstmal einen Einblick in die Entwicklung mit JavaME erhalten und habe vieles über die Standard-Entwicklung nach dem Java Community Process gelernt. Ein weiteres mal habe ich auch gesehen wie wichtig vorausgehende Planung und ein gutes Design sind.

Die Zusammenarbeit mit der cnlab AG war, wie bereits während der Semesterarbeit, immer sehr angenehm. Die Sitzungen konnten meist speditiv abgehalten werden und brachten uns forwärts. Mit René Vogt hatten wir auch einen sehr kompetenten Partner, der bei technischen Fragen immer sehr gut Antwort geben konnte.

Über alles würde ich die Arbeit als Erfolg werten. Wir haben unser Ziel, die bestehende Anwendung mit JavaME nachzubauen und einige Teile zu verbessern und zu erweitern mit nur kleinen Einschränkungen erreicht und ich habe in diesen 16 Wochen sehr viel lernen und auch viele Erfahrungen sammeln können.

E.2 Michael Wagner

Mit den Erfahrungen aus der Studienarbeit im letzten Semester ging die Bachelorarbeit einiges reibungsloser über die Bühne. Dazu beigetragen hat sicher auch die Tatsache, dass

wir wieder mit der gleichen personellen Besetzung im Projektteam, in der Projektbetreuung und beim Industriepartner arbeiteten.

Obwohl sich die Bachelorarbeit im gleichen Themengebiet wie die Studienarbeit bewegte, war sie aus technologischer Sicht etwas komplett anderes. In der Studienarbeit beschäftigten wir uns mit Ruby on Rails mit der serverseitigen Auswertung der Daten. Die Bachelorarbeit beschäftigte sich nun mit JavaSE und JavaME und der clientseitigen Datensammlung. Die Entwicklung von Software für kleine mobile Geräte in JavaME war für mich Neuland.

Das Entwickeln mit JavaME barg viele, meist negative Überraschungen. Da waren zuerst einmal die sehr stark beschränkten Ressourcen der mobilen Geräte. Dazu kam, dass viele Klassen und Methoden wie z.B. String.format() die in JavaSE vorhanden sind, in JavaME nicht existieren und selbst programmiert werden mussten. Zusätzlich erschwerend war die Tatsache, dass die JavaME Klassen unter Java 1.4 kompiliert werden müssen und so viele angenehme Programmiertechniken wie z.B. Generics nicht vorhanden sind.

Ein weiterer spannender Teil der Bachelorarbeit war der Zugriff auf die OBD-II-Schnittstelle von PKWs. Es zeigte sich, dass unterschiedliche Fahrzeuge trotz einer standardisierten Schnittstelle sehr unterschiedliches Verhalten zeigten. So gab es grosse Unterschiede bei den Daten, welche die Fahrzeuge liefern, und beim zeitlichen Verhalten (Reaktionszeiten). Es war eine grosse Herausforderung, eine Software zu entwickeln, welche mit dieser Heterogenität umgehen konnte, um möglichst viele Fahrzeuge zu unterstützen.

Rückblickend habe ich sehr viel über Software-Entwicklung auf verschiedenen Plattformen gelernt und wertvolle Erfahrungen gesammelt. Auch habe ich die Wichtigkeit von Prototyping im Vorfeld einer Software-Entwicklung erkannt. Prototyping kann schon sehr früh Aufschluss über die Machbarkeit und Möglichkeiten liefern und so das Risiko späterer Überraschungen reduzieren.

Ich hoffe, dass ich das während der Bachelor-Arbeit angeeignete Wissen und die Projekterfahrung nun in meiner beruflichen Tätigkeit nach dem Studium einsetzen kann.

Nomenclature

ant	Das in Java geschriebene Open Source Tool Ant dient der automatisierten Programmerzeugung aus dem Source Code. Seine Aufgabe entspricht somit dem besonders unter Linux verbreiteten make.
GPRS	General Packet Radio Service
GPS	Global Postitioning System: Satelitten basiertes Positionierungs-System
J2ME	Java Micro Edition: Java Applikationsplattform für Mobile Geräte
JSR	Java Specification Request
MIDP	Mobile Information Device Profile
MSA	Mobile Service Architecture
OBD-II	On Board Diacnostic Interface 2: Schnittstelle um diverse Parameter von einem Fahrzeug zu lesen/schreiben.
PID	OBD-II Parameter ID: Definiert welche Nachricht an den OBD Controller gesendet werden soll und wie die Nachricht aussieht
Ruby	Ruby ist eine moderne, vielseitige Programiersprache die Mitte der Neunziger Jahre vom Japaner Yukihiro Matsumoto entworfen wurde.
VSZ	Verkehrssicherheitszentrum: Im Zusammenhang mit dieser Bachelorarbeit ist immer das TCS Verkehrssicherheitszentrum Betzholz in Hinwil gemeint.
WAB2	2. Phase der 2-Phasen-Weiterausbildungskurse für den Führerausweis in der Schweiz.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Apache Foundation. Apache license, version 2.0. http://www.apache.org/licenses/LICENSE-2.0.html, Jan 2004. 3.8.1, 3.8.2, 3.8.3, 3.8.4

[2] Apache Foundation. Log4j. http://logging.apache.org/log4j/, May 2009. 3.8.1

[3] Athlosoft. athlosoft, 2009. 2.6.2

[4] Becke, M., Dever, J., dIon Gillard, Glueck, O., Sung-Gu, Kalnichevski, O., Sullivan, S. C., Sutton, A., and Waldhoff, R. Apache commons httpclient. http://hc.apache.org/httpclient-3.x/, Aug 2007. 3.8.4

[5] CarCode MÃijller. Elmcan-elm327. http://www.obd-2.de/, Mar 2009. 2.2

[6] cnlab AG. cnlab ag - information technology research. http://www.cnlab.ch/. 2.6.2

[7] Delagrange, M., Waldhoff, R., McClanahan, C., Sanders, S., Donkin, R. B., Donald, P., Manolache, C., Sitze, R., Baliuka, J., Kitching, S., Lundberg, D., and Stansberry, B. Apache commons logging. http://commons.apache.org/logging/, Nov 2007. 3.8.4

[8] Dr.med. Friederike Bischof, MPH. Physical activity level, 2009. 2.3, E.2

[9] ESA. EuropÃd'ische weltraumorganisation. http://www.esa.int, 2009. 2.2, E.2

[10] Estey, L. Unavco, 2009. 2.1.1

[11] Free Software Foundation. Gnu lesser general public license. http://www.gnu.org/licenses/old-licenses/lgpl-2.1.txt, Jan 1999. 3.8.2, 3.8.5, 3.8.6

[12] Free Software Foundation. Gnu public license. http://www.gnu.org/licenses/gpl.html, Jun 2007. 3.8.3

[13] Geospatial Science Division. World geodetic system 1984. http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html, 1997. 2.1, 2.1, 2.1.1

[14] Gilbert, D. Jfreechart. http://www.jfree.org/jfreechart/, Apr 2009. 3.8.5

[15] Halliday, S. Openlapi. http://code.google.com/p/openlapi/, Jun 2008. 3.8.6

[16] JFlex. Jflex - the fast scanner generator for java, 2009. 3.5.1

[17] Karlsson, J., Katz, D., Ohme, K., and Larne, H. Microlog. http://sourceforge.net/projects/microlog, May 2009. 3.8.1

[18] Loytana, K., and Nokia Corporation. Location api. http://jcp.org/en/jsr/detail?id=179, Mar 2006. 2.1, 2.6.5

[19] Loytana, K., and Nokia Corporation. Mobile media api. http://jcp.org/en/jsr/detail?id=135, Mar 2009. 2.6.4

[20] mobimotion GmbH. mobimotion spurty-brustgurt. http://www.mobimotion.com/index.php?id=brustgurt, Mar 2009. 2.6.2, A.2.2.4

[21] NASA. Earth's gravity definition. `http://www.csr.utexas.edu/grace/gravity/gravity_definition.html`, 2004. 2.1.1

[22] NASA. Nasa goddard space flight center. `http://www.nasa.gov/centers/goddard/home/`, 2008. 2.1.1

[23] National Geospacial-Intelligence Agency. National imagery and mapping agency. `https://www1.nga.mil/Pages/Default.aspx`, 2008. 2.1.1

[24] Niemela, P., and Nokia Corporation. Mobile sensor api. `http://jcp.org/en/jsr/detail?id=256`, Dec 2008. 2.6.3, 4.2

[25] Nokia. Nokia sports tracker, 2009. 2.6.2

[26] Nokia Corporation. Midp 2.1 api access rights. `http://wiki.forum.nokia.com/index.php/MIDP_2.1_API_access_rights`, Jun 2007. 2.9.1, 2.9.1

[27] SBB. Navigo. `http://www.sbb-mobileworld.ch/fahrplan/navigo/`, March 2009. 2.8

[28] Sony Ericsson. Bluetooth gps-pulsgurt, 2009. 2.6.2

[29] Sun Microsystems. Sun. `http://www.sun.com`, Mar 2009. 2.8

[30] Thompson, T., and Motorola. Java api for bluetooth. `http://jcp.org/en/jsr/detail?id=82`, Aug 2008. 2.6.1

[31] Totterman, P., and Skarzhevskyy, V. Bluecove jsr-82 project. `http://www.bluecove.org/`, Jun 2009. 3.8.3

[32] Turner, S. Jmicropolygon. `http://sourceforge.net/projects/jmicropolygon/`, May 2009. 3.8.2

[33] Wikipedia. Harris benedict equation, 2009. 2.5.1

[34] wikipedia.org. Gemeinfreiheit. `http://de.wikipedia.org/wiki/Gemeinfreiheit`, 2009. 3.1, E.2

[35] wikipedia.org. Obd-ii pids. `http://en.wikipedia.org/wiki/OBD-II_PIDs`, Mar 2009. 2.2.1

[36] Yandell, H., OBrien, T., Sanders, S., Waldhoff, R., Rall, D., Stevens, J. S., Gregory, G. D., and Graham, D. Apache commons codec. `http://commons.apache.org/codec/`, Jul 2004. 3.8.4

1

[1]Die am Ende einer Zeile aufgeführten Nummern sind die Kapitelnummern, welche die Quellenangaben jeweils betreffen.