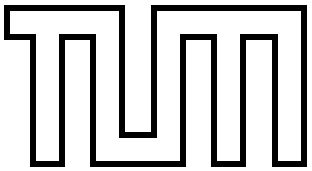


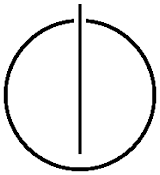
FAKULTÄT FÜR INFORMATIK

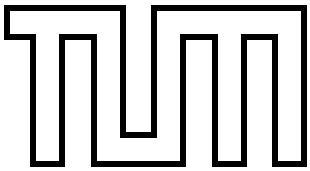
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Konzeption und Realisierung eines mobilen verteilten Systems zur Aggregation und Analyse von Echtzeitdaten eines Versuchsfahrzeugs

Sebastian Gallenmüller





FAKULTÄT FÜR INFORMATIK

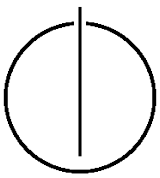
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Konzeption und Realisierung eines mobilen verteilten Systems zur Aggregation und Analyse von Echtzeitdaten eines Versuchsfahrzeugs

Conception and realization of a mobile distributed system for aggregation and analysis of real time data of a prototype car

Bearbeiter:	Sebastian Gallenmüller
Aufgabensteller:	Prof. Dr. Uwe Baumgarten
Betreuer:	Nils Kannengießer, M. Sc.
Abgabedatum:	15. März 2012



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 14. März 2012 Sebastian Gallenmüller

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben.

Danken möchte ich meinem Betreuer Herrn Kannengießer für seine Geduld beim Finden der Themenstellung, sowie für seine Verbesserungsvorschläge. Ein herzliches Dankeschön an Prof. Baumgarten, der diese Arbeit erst ermöglicht hat. Auch ein Danke an beide für den Kauf der Hardware.

Mein besonderer Dank gilt auch dem TUfast Eco-Team für die gute Zusammenarbeit und besonders David Kalwar, der die Idee zu dieser Arbeit hatte.

Ein Dankeschön auch an die Korrekturleser Paul Emmerich, Ursula Gallenmüller und Roland Röger.

Kurzfassung

Im Rahmen des Shell Eco-marathons entwickelt ein Team von Studenten an der TU München ein Fahrzeug, das auf maximale Treibstoffeffizienz hin optimiert wird, was nicht zuletzt durch ein möglichst kleines und leichtes Fahrzeug erreicht werden kann. Da heutige Smartphones bereits eine Vielzahl von Sensoren und Kommunikationsmöglichkeiten bei gleichzeitig sehr geringem Gewicht und Platzbedarf integriert haben, bietet es sich an, die Funktionen eines Smartphones für dieses Fahrzeug zu nutzen.

Für diese Aufgabe wurde in der vorliegenden Arbeit ein System konzipiert und realisiert, das die Ansteuerung, die Anzeige der Daten und deren Aufzeichnung für das Fahrzeug übernimmt. Dieses System gliedert sich in folgende drei Teilsysteme:

Teilsystem eins besteht aus einem *Controllerboard*, für das eine Software entwickelt wird, um die angeschlossene Aktorik und Sensorik des Fahrzeugs anzusteuern. Bei der verwendeten Hardware handelt es sich um das Accessory Development Board, das mit einem Android Smartphone verbunden wird.

Das zweite Teilsystem ist ein Android *Smartphone*, das eine App erhält, sodass es einerseits als Instrumententafel fungiert, die dem Fahrer Informationen wie Geschwindigkeit oder Batterieladestand anzeigt, andererseits dient es auch zur Sprachkommunikation mit der Box während eines Rennens und zur Übermittlung von Daten der Sensorik an einen Webservice.

Drittes Teilsystem ist ein *Webservice*, der geplant und implementiert wird. Der Webservice empfängt die Daten des Smartphones, speichert sie und stellt sie über ein Webinterface zur Verfügung. Dieser Webservice wird mit Hilfe der Google App Engine im Zusammenspiel mit dem Google Web Toolkit realisiert.

INHALTSVERZEICHNIS

Teil I.

Einführung

KAPITEL

EINS

VORWORT

Im Rahmen dieser Bachelorarbeit wurde ein System für ein Fahrzeug entwickelt, das an einem Effizienzwettbewerb, dem Shell Eco-marathon teilnehmen wird. Der Wettbewerb wird bereits seit 1985 jährlich veranstaltet. Bei diesem Rennen gewinnt das Fahrzeug, welches die maximale Reichweite aus einem Liter Benzin oder einer äquivalenten Menge eines alternativen Energieträgers gewinnen kann [36].

Das TUfast Eco-Team der TU München nahm 2011 mit einem selbst entwickelten Fahrzeug, dem Htu 11, zum ersten Mal an diesem Wettbewerb teil. Das Fahrzeug besaß einen Wasserstoffantrieb. Da sich diese Form des Antriebs als relativ unzuverlässig erwiesen hat, wurde entschieden, das bisherige Fahrzeug auf Batteriebetrieb umzurüsten. Mit diesem neuen Antrieb und mit weiteren Verbesserungen, die zum Teil im Rahmen dieser Arbeit entwickelt wurden, ist geplant, dass das Fahrzeug, nun als eLi 12, vom 17. - 19. Mai 2012 in Rotterdam wieder am Wettbewerb teilnimmt.

Abbildung 1.1 zeigt eine Konzeptgrafik des eLi 12. Das Chassis selbst besteht größtenteils aus Kohlefaser und bietet Platz für einen Fahrer. Nach den beschriebenen Umbauten, soll das Fahrzeug bei Fertigstellung ca. 30 kg ohne Fahrer wiegen. Die erreichbare Geschwindigkeit soll dann ca. 30 km/h betragen, womit bei konservativer Berechnung eine Reichweite von ca. 2000 km/l erreicht werden kann.

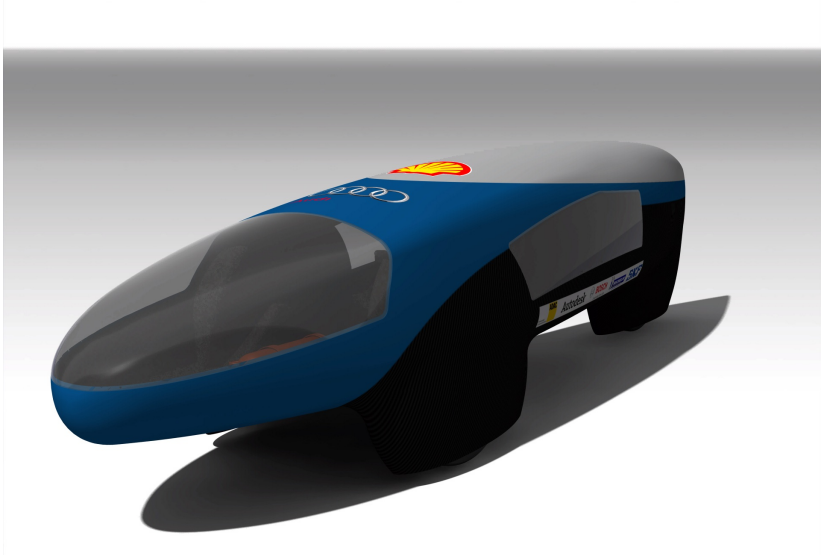


Abbildung 1.1.: Darstellung des eLi 12

KAPITEL

ZWEI

EINLEITUNG

Heutige Smartphones bieten eine Vielzahl von Kommunikationsmöglichkeiten und Sensoren auf sehr geringem Raum zu erschwinglichen Preisen. Durch die mittlerweile ausgereiften Programmierumgebungen für diese Mobiltelefone können die Funktionen von jedem Entwickler für eigene Applikationen genutzt werden. Im Zuge dieser Bachelorarbeit sollen diese Funktionen für ein Fahrzeug, das ein Team von Studenten der TU München entwirft und fertigt, nutzbar gemacht werden.

Das Fahrzeug erhält ein Controllerboard, an das die Sensoren des Fahrzeugs, die Motorsteuerung und das Batteriemanagementsystem angeschlossen werden. Dieses Bauteil stellt die gewünschten Daten dem Smartphone zur Verfügung. Dazu bietet sich das sogenannte Android Open Accessory Development Kit an, ein Board, das per USB mit einem Android Smartphone verbunden werden kann und durch einfaches Einbinden einer Bibliothek auf einem Android Mobiltelefon in eigenen Applikationen verwendet werden kann. Dieses Board ist auch der Grund, warum Android für dieses Projekt verwendet wird [16].

Das Android Smartphone wird für mehrere Aufgaben verwendet. Es erhält die Informationen vom Controller und sammelt über einen im Telefon integrierten GPS Sensor selbst Positionsdaten. Daneben wird es als Instrumentenanzeige für das Fahrzeug und zur Sprachkommunikation mit dem Fahrer verwendet. Es dient aber auch dazu, die gesammelten Informationen zwischenzuspeichern und an einen Webserver zu übermitteln.

Für den Webserver soll schließlich eine Software entwickelt werden, die die gesammelten Daten entgegennimmt, speichert, für die Nutzer aufbereitet und zugänglich macht. Realisiert wird der Webservice mit App Engine, einem Cloud Service von Google. Die Nutzerschnittstelle wird mit Google Web Toolkit realisiert, das eine einfache Erstellung von dynamischen Webseiten in Java ermöglicht.

Im Rahmen dieser Arbeit wird jeweils die Software für Controller, Smartphone und Server entwickelt. Zunächst soll dazu ein Überblick über das gesamte System gegeben werden, in dem auf die allgemeine Problembeschreibung, die Anforderungen und den grundlegenden Aufbau des Systems eingegangen wird. Anschließend wird jedem Teil des Sys-

tems - dem Controller, dem Smartphone und dem Server - ein eigenes Kapitel gewidmet, in dem jeweils die Umgebung, die Bedienung, ggf. die Nutzeroberfläche und schließlich der innere Aufbau des Systems beschrieben wird. Abgeschlossen wird die Arbeit mit einem Kapitel, das das Testen des fertigen Systems beschreibt und die Ergebnisse noch einmal zusammenfasst.

Teil II.

Systemüberblick

KAPITEL

DREI

PROBLEMSTELLUNG

Dieses Kapitel beschreibt die Problemstellung und typische Anwendungsfälle für das System, aus denen im folgenden Kapitel Anforderungen an das System abgeleitet werden.

3.1. Problembeschreibung

Für den Shell Eco-marathon 2012 soll ein neues Fahrzeug entwickelt werden. Als Grundlage dient das Fahrzeug von 2011, von dem das Chassis weiter verwendet wird, die meisten anderen Komponenten aber neu entwickelt, gebaut und ausgetauscht werden. Unter anderem wird der Wasserstoffantrieb, der sich als relativ unzuverlässig erwiesen hat, gegen einen Antrieb mit Batterien als Speichertechnik ausgetauscht.

Das bisherige System besteht aus einem Controller, der die Brennstoffzelle steuert und einer Motorsteuerung, die den Elektromotor regelt. Die Motorsteuerung hängt wiederum an einem Controller, der Daten wie aktuelle Stromstärke und Geschwindigkeit aus der Motorsteuerung ausliest und dem Fahrer auf einem Display anzeigt. Eine Aufzeichnung der gesammelten Daten findet nicht statt.

Beim Fahrzeug für das Rennen 2012 wird die Brennstoffzelle und der dazugehörige Controller gegen ein Batteriemanagementsystem und Akkus getauscht. Motor und Motorsteuerung werden weiterverwendet. Neben der Umrüstung auf einen anderen Antrieb, sollen Daten zur späteren Analyse für die Entwicklung künftiger Fahrzeuge aufgezeichnet werden. Die Daten sollen aber auch zur Überwachung des Fahrzeugs während des Rennens eingesetzt werden, um eventuelle Probleme frühzeitig zu erkennen. Dafür sollen die Daten während der Fahrt zusammen mit dem aktuellen Aufenthaltsort an einen Server übermittelt, live dargestellt und dort auch für eine spätere Analyse aufbewahrt werden.

Das Ziel beim Shell Eco-marathon ist es ein Fahrzeug zu entwickeln, das eine geforderte Strecke mit möglichst geringem Energieverbrauch zurücklegen kann. Bei der Berechnung dieses Verbrauches spielt der Verbrauch der Elektronik selbst keine Rolle, da lediglich der Verbrauch des Motors gemessen wird. Wichtig ist vor allem das Gesamtgewicht des Fahr-

zeugs so niedrig wie möglich zu halten. Da der Fahrer während des Rennens ohnehin ein Mobiltelefon für die Kommunikation mit der Box verwendet, bietet es sich an, die Funktionen des Mobiltelefons zu nutzen, also Sensorik und Kommunikationsmöglichkeiten, da hierbei die gewünschten Funktionen ohne eine Erhöhung des Gewichts realisiert werden können. Zusätzlich kann das Mobiltelefon als Instrumententafel dienen und so das bisherige Display ersetzen.

3.2. Anwendungsfälle

Im folgenden Kapitel sollen einige typische Einsätze des Systems beschrieben werden. Diese sogenannten Anwendungsfälle beschreiben das Verhalten eines Systems, das von außen beobachtbar ist. Beobachter sind hierbei die Akteuere, die entweder Personen oder Systeme sein können. Im vorliegenden Beispiel gibt es zwei Rollen für Personen, den Fahrer des Fahrzeugs und den Mechaniker, der das Renngeschehen von der Boxengasse aus beobachtet. Daneben gibt es noch drei verschiedene Systeme, den Controller, das Smartphone und einen Webserver [7, S. 69ff].

Smartphoneeinstellung

Akteure	*Fahrer, Smartphone, Mechaniker, Webservice*
Beschreibung	Der *Fahrer* ruft den Einstellungsdialog des *Smartphones* auf. Das *Fahrer* aktiviert am *Smartphone* den Datentransfer zum *Webservice*. Der *Fahrer* stellt die Telefonnummer des *Mechanikers* am *Smartphone* ein. Der *Fahrer* gibt die Dauer des Rennens, die Strecke und die Rundenanzahl am *Smartphone* ein.
Vorbedingung	Smartphone ist noch nicht eingestellt.
Nachbedingung	Smartphone ist eingestellt.

Sprachkommunikation

Akteure	*Fahrer, Smartphone, Mechaniker*
Beschreibung	Der *Fahrer* wählt mit Hilfe der Bedienelemente am Lenkhebel die Telefonfunktion. Das *Smartphone* wählt die voreingestellte Telefonnummer der Box. Der *Mechaniker* nimmt den Anruf an und kann mit dem *Fahrer* sprechen. Der *Fahrer* passt mit Hilfe der Bedienelemente die Lautstärke an.
Vorbedingung	Smartphone ist bereits eingestellt.
Nachbedingung	Telefonverbindung aufgebaut.

Steuerung des Fahrzeugs

Akteure	*Fahrer, Controller, Smartphone*
Beschreibung	Der *Fahrer* steigt in das Fahrzeug ein. Zum Entsperren des Fahrzeugs drückt er eine bestimmte Tastenkombination an den Bedienelementen am Lenkhebel. Der *Controller* erkennt das Entsperrsignal und gibt die anderen Bedienelemente frei. Der *Fahrer* kann nun mit Hilfe der Bedienelemente das Fahrzeug starten, beschleunigen oder abbremsen. Der *Controller* sammelt während der Fahrt Daten von den angeschlossenen Sensoren und schickt diese an das *Smartphone*. Das *Smartphone* zeigt auf seinem Display aktuelle Fahrinformationen für den *Fahrer* an.
Vorbedingungen	Fahrzeug ist noch nicht gestartet Smartphone ist bereits eingestellt.
Nachbedingungen	Fahrzeug fährt und Smartphone zeigt Fahrdaten an.

Datenaufzeichnung und Kommunikation

Akteure	*Controller, Smartphone, Webservice*
Beschreibung	Der *Controller* schickt Daten zum *Smartphone*. Das *Smartphone* sammelt Positionsdaten mit Hilfe des eingebauten GPS Sensors. Das *Smartphone* speichert die gesammelten Information zwischen. Das *Smartphone* sendet mehrmals pro Minute die gesammelten Daten an den *Webservice*. Der *Webservice* speichert die empfangenen Daten dauerhaft und zeigt diese auf einem Browser an.
Vorbedingungen	Smartphone ist bereits eingestellt.
Nachbedingungen	Fahrdaten befinden sich auf dem Webserver.

KAPITEL

VIER

ANFORDERUNGEN

Im vorliegenden Kapitel sollen die wichtigsten Anforderungen an das System vorgestellt werden. Dabei werden Anforderungen in zwei Kategorien von Funktionen unterschieden: funktionale Anforderungen, die eine konkrete Funktion eines Systems darstellen, und nichtfunktionale Anforderungen, die Einschränkungen darstellen, die nicht unmittelbar auf eine Funktion des Systems bezogen sind [7, S. 38].

Zusätzlich zu den bereits genannten Kategorien werden auch noch die Anforderungen, die sich aus den Regeln des Shell Eco-marathons ergeben, unter Nebenbedingungen aufgeführt.

Die einzelnen Funktionen sind jeweils in absteigender Wichtigkeit sortiert.

4.1. Funktionale Anforderungen

- **Datenerfassung und Anzeige:** Fahrdaten sollen mit Hilfe des Controllers (von Motorsteuerung, Batteriemanagement und angeschlossener Sensorik) erfasst und auf einem Smartphone angezeigt werden.
- **Datenarchivierung:** Die empfangenen Daten des Controllers sollen auf dem Smartphone um die Ortungsinformation aus dem GPS Sensor erweitert und zu einem Webservice übermittelt werden.
- **Datendarstellung:** Die vom Webservice archivierten Daten sollen mit Hilfe eines Browsers dargestellt werden. Dabei sollen die Daten auf einer Landkarte oder als Tabelle dargestellt werden können.
- **Sprachkommunikation:** Das Smartphone soll während der Fahrt eine Kommunikation mit den Mechanikern in der Box ermöglichen und dabei weiterhin als Anzeige für Fahrzeugdaten dienen.
- **Einstellungsmöglichkeit:** Am Smartphone sollen Parameter wie Dauer des Rennens, Länge der Rennstrecke und Rundenanzahl einstellbar sein.

4.2. Nichtfunktionale Anforderungen

- **Verfügbarkeit:** Es soll ein möglichst ausfallsicheres System entstehen, sodass das Fahrzeug auch bei Ausfall des Mobiltelefons steuerbar bleibt. Dazu muss der Fahrer das Fahrzeug auch steuern können, ohne dass das Smartphone mit dem Controller verbunden ist.
- **Bedienbarkeit:** Da der Fahrer während des Rennens Handschuhe trägt, ist eine Bedienung des Smartphones per Touchscreen nicht möglich. Deshalb müssen während der Fahrt relevante Funktionen auch ohne Touchscreen ausführbar sein. Dazu werden die Bedienelemente, die am Lenkhebel befestigt sind, benutzt.
- **Leistung:** Die Übertragung der Daten an den Webserver soll alle 10 Sekunden stattfinden, um so während des Rennen stets möglichst aktuelle Daten vorliegen zu haben. Ferner soll spätestens alle 5 Sekunden eine Messung vorgenommen werden, deren Ergebnisse zum Server geschickt werden. Die Anzeige für den Fahrer sollte mehrmals pro Sekunde aktualisiert werden.
- **Zuverlässigkeit:** Bei einer Nichterreichbarkeit der Servers, z. B. aufgrund einer unzuverlässigen Datenverbindung, sollen die Daten nicht gelöscht, sondern zum nächstmöglichen Zeitpunkt zum Server übertragen werden.

4.3. Nebenbedingungen

- Laut Artikel 16 des Reglements darf der Fahrer nur Kommunikationsgeräte einsetzen, die er freihändig bedienen kann. Das heißt, das Smartphone darf während der Fahrt vom Fahrer nicht per Touchscreen bedient werden [37, Art. 16].
- Für die Versorgung der Fahrzeugelektronik ist genau eine Hilfsbatterie erlaubt, die selbst allerdings keine Energie für den Antrieb zur Verfügung stellen darf. Das heißt, die Energie aus dem Akku des Smartphones darf nur das Telefon betreiben und nicht anderweitig genutzt werden in [37, Art. 57, g-k].

KAPITEL

FÜNF

GRUNDLEGENDER AUFBAU DES SYSTEMS

Bereits in den Anwendungsfällen wird das Gesamtsystem in drei Teilsysteme aufgeteilt, den Controller, das Smartphone und den Webservice. Da es sich bei diesen drei Systemen um unabhängige Systeme handelt, die miteinander über ein Netzwerk kommunizieren, liegt ein verteiltes System vor. Im Gesamtsystem, das nun entwickelt werden soll, kommt noch hinzu, dass zwei der Systeme, als Teil eines Fahrzeugs, mobil sein werden. Deshalb handelt es sich bei diesem System, um ein sogenanntes mobiles verteiltes System [28, S. 16ff].

Da für dieses Projekt ein Smartphone mit einem Controllerboard verbunden werden soll, bietet es sich an ein Android Smartphone zu verwenden. Denn für diese Plattform wurde im März 2011 das Open Accessory Development Kit vorgestellt. Dieses Kit besteht aus einem Hardwarecontroller, der per USB mit dem Smartphone verbunden werden kann. Sowohl für den Controller als auch für das Smartphone werden Bibliotheken bereitgestellt, die eine schnelle und einfache Verbindung dieser zwei Systeme ermöglichen[16].

Für den Controller wird eine Software entwickelt, die die Daten der Motorsteuerung, des Batteriemanagementsystems und weiterer angeschlossener Sensoren erhält. Ebenfalls werden an dieses Board die Bedienelemente, die sich auf dem Lenkhebel des Autos befinden angeschlossen. Die Software soll nun diese Daten aufbereiten und mittels eines Protokolls, das ebenfalls entwickelt werden muss, auf dem USB Port ausgeben. Den genauen Aufbau dieses Systems beschreibt Kapitel III.

Auf Seiten des Smartphones muss ebenfalls eine App entwickelt werden, die zunächst einmal die Daten, die es vom Controller bekommt auswertet, filtert, darstellt und aufzeichnet. In diesen Daten sind auch die Statusmeldungen der Bedienelemente enthalten, mit deren Hilfe nicht nur das Fahrzeug, sondern auch das Programm bedient wird. Daneben zeichnet die App mit Hilfe des im Telefon eingebauten Sensors die GPS Daten auf. Neben der Oberfläche, die die Fahrdaten anzeigen soll, muss die Applikation auch einen Einstellungsdialog bieten, auf dem die geforderten Einstellungen vorgenommen werden können. Daneben muss die Anwendung noch die Funktion bieten sowohl Fahrzeugdaten an den Webservice als auch Sprachdaten an die Mechaniker in der Box zu senden. Eine

detaillierte Beschreibung von Umsetzung und Funktion dieser Applikation findet sich in Kapitel IV wieder.

Für den Webservice wird die Goole App Engine verwendet. Ein Service, bei dem ein Programm eine Applikation auf einer von Google bereitgestellten Infrastuktur ausgeführt wird. Ein Vorteil dieser Lösung ist, dass hierbei keine Administrations- und Konfigurationsarbeiten anfallen, da die Infrastruktur von Google gewartet und konfiguriert wird. Ein weiterer Vorteil ist die Flexibilität dieser Lösung, sodass ein gewisses Kontingent, wie z. B. ein Gigabyte Übertragungsvolumen pro Tag kostenlos ist, aber bei Bedarf weitere Kapazitäten gebucht werden können [24, 27].

Der Webservice muss ebenfalls mehrere Funktionen erfüllen. Zunächst dient er als Datenbank für die Aufzeichnung der Fahrzeugdaten. Dazu muss er eine Schnittstelle für die Android App bereitstellen, über die die Fahrzeugdaten angeliefert werden. Auf der anderen Seite soll er aber auch eine Schnittstelle für eine Webseite bieten, mit deren Hilfe zum einen die Live Überwachung der aktuellen Fahrt möglich ist. Zusätzlich sollen aber auch die archivierten Daten visualisiert werden können z. B. mit Hilfe einer Tabelle, eines Graphen oder auf einer Landkarte. Die konkrete Implementierung dieses Webservices wird in Kapitel V vorgestellt.

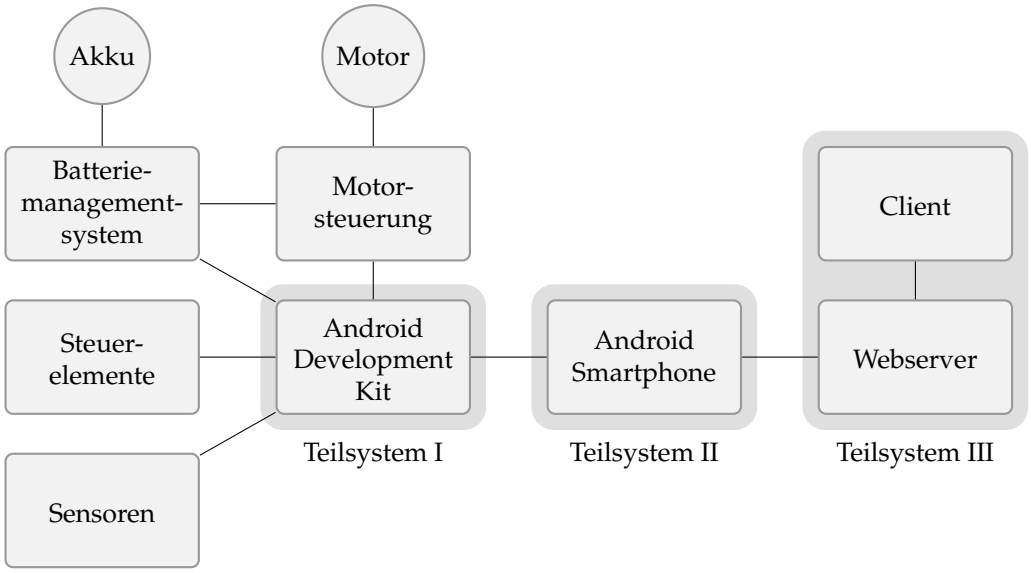


Abbildung 5.1.: Schematische Darstellung des Gesamtsystems

Abbildung 5.1 zeigt wie das Gesamtsystem aus den Teilsystemen aufgebaut ist, die in dieser Bachelorarbeit entwickelt werden.

Teil III.

Controller

KAPITEL

SECHS

HARDWARE

In diesem Kapitel werden die Hardware des Controllers, weitere Hardwarekomponenten, die direkt mit dem Controller in Verbindung stehen und der Schaltplan näher vorgestellt.

6.1. Hardwarekomponenten

Es folgt eine Auflistung der wichtigsten verwendeten Hardwarekomponenten.

Controller

Das Arduino Mega Accessory Development Kit ist ein Board, das als Teil einer Familie von Open Source Entwicklungsboards entworfen wurde, die den Namen Arduino trägt. Das Board basiert auf dem Prozessor ATmega2560, einer 8 Bit CPU, die mit 16 MHz getaktet wird. Wie alle Arduino Boards besitzt es einen USB Port, über den neue Software auf den Controller geladen wird. Zusätzlich kann der Controller über diesen Port mit Strom versorgt werden, allerdings ist bei dieser Art der Stromversorgung die 5 V Spannung, die das Board für Sensoren bereitstellt instabil. Dieses Problem kann dadurch behoben werden, dass das Board idealerweise mit einer Spannung von 7 - 12 V versorgt wird. Die Hilfsbatterie im Fahrzeug liefert eine Spannung von 12 V, wodurch die Sensoren mit einer zuverlässigen 5 V Spannung vom Controller versorgt werden können.

Eine Besonderheit dieses Boards ist der zweite USB Anschluss. Dieser dient dazu, eine Verbindung zum Mobiltelefon aufzubauen, zusätzlich kann das Telefon über diesen Port auch mit Strom versorgt werden, sodass der Akku des Smartphones während des Rennens geladen werden kann.

Auf dem Board befinden sich zudem noch 16 analoge Eingänge, an denen später die analogen Sensoren angeschlossen werden. Diese Eingänge besitzen eine Auflösung von 10 Bit, d. h. ein anliegendes Signal kann in genau 1024 Stufen unterschieden werden.

Neben den analogen existieren auch 54 digitale Eingänge, an die später die Bedienelemente angeschlossen werden. Insgesamt sechs dieser Ein-/Ausgänge bieten die Zusatzfunktion an, sie für einen seriellen Anschluss (jeweils zwei Pins) zu nutzen. An einen dieser Anschlüsse wird z. B. die Motorsteuerung angeschlossen. Weitere sechs Anschlüsse sind interruptfähig, d. h. ein eintreffendes Signal an einem dieser Eingänge kann sehr schnell erkannt und speziell behandelt werden. Aufgrund der Knappheit der Interrupteingänge können nicht alle verfügbaren Bedienelemente an einen Interruptport angeschlossen werden, sondern nur wichtige wie z. B. die Bremse. Zusätzlich werden zwei dieser Ports vom Anschluss des Batteriemanagementsystems belegt, da diese Pins für das Two Wire Interface benötigt werden. Die anderen Bedienelemente werden regelmäßig abgefragt und verwenden nur jeweils einen der digitalen Eingänge [1].

Batteriemanagementsystem

Das eingesetzte Batteriemanagementsystem verwendet den Controller OZ890 des Unternehmens O2Micro. Aufgabe des Controllers ist es die 13 Zellen des Lithium-Ionenakkus im Fahrzeug zu überwachen. Dabei schützt das Batteriemanagement den Akku vor Über- und Unterspannung, Überlastung, Kurzschluss, Überhitzung und Unterkühlung. Außerdem gleicht er die Spannungen der einzelnen Akkuzellen einander an. Neben den Schutzfunktionen bietet der OZ890 auch noch ein Two Wire Interface zur Abfrage von Messdaten [31].

Für dieses Projekt werden die einzelnen Zellspannungen ausgelesen, um die Gesamtspannung zu ermitteln und um eventuell beschädigte Zellen frühzeitig zu erkennen und gegebenenfalls austauschen.

Motorsteuerung

Bei der Motorsteuerung handelt es sich um das Produkt Solo Whistle des Unternehmens Elmo Motion Control. Die Aufgabe der Motorsteuerung ist es die Strommenge, die zum Motor geliefert wird, zu regeln. Angebunden wird die Motorsteuerung über ihren integrierten RS232 Anschluss an das Arduino ADK Board. Für die Verbindung von serieller Schnittstelle des Arduino mit dem RS232 wird ein MAX232 Transceiver benutzt [12, 29].

Neben der Regelungsfunktion kann bei der Steuerung auch die Drehzahl des Motors abgefragt werden, mit deren Hilfe die Geschwindigkeit des Fahrzeug errechnet wird.

Temperatursensor

Beim Temperatursensor handelt es sich um einen LM35 CZ von Texas Instruments. Dieser Sensor besitzt drei Pins. Zwei für die Stromversorgung, die mit dem 5 V und GND Pins des Arduino ADK Boards verbunden werden, und einen analogen Ausgang, der an einen Analogpin des Controllers angeschlossen wird. An diesem analogen Pin kann die Temperatur ausgelesen werden, da pro 10 mV pro Grad Celsius an diesem Ausgang anliegen. Der Messbereich des Sensors liegt in der verwendeten Konfiguration zwischen +2° und

+150° Celsius [38].

Dieser Messbereich ist ausreichend, da der Sensor verwendet wird, um die Motortemperatur zu überwachen, um so eine Überhitzung des Motors frühzeitig zu erkennen. Dabei wird die Temperatur des Motors voraussichtlich 50° Celsius nie übersteigen.

Stromsensor

Der verwendete Stromzähler ist ein ACS715, der den Strom im Bereich von -1,5 A bis 30A messen kann. Wie der Temperatursensor, besitzt auch dieser drei Pins, wobei zwei für die Spannungsversorgung (5V) genutzt werden und der weitere Pin den gemessenen Stromverbrauch repräsentiert. Es handelt sich hierbei ebenfalls um einen analogen Sensor, d. h. am analogen Ausgang des Sensors liegt der gemessene Strom als Wert zwischen 0 V und 5 V an, wobei pro gemessenem Ampere jeweils 0,133 mV anliegen. Dabei ist allerdings zu beachten, dass der Sensor einen verschobenen Messbereich besitzt, d. h. 0 V am analogen repräsentieren -1,5 A; 0,5 V bedeuten, dass 0 A vom Sensor gemessen werden [32].

Der Stromsensor wird verwendet, um den Stromverbrauch des Motors zu messen. Dabei wird die maximale Stromstärke voraussichtlich 10 Ampere betragen. Der Stromsensor mit 30 A ist dafür ausreichend dimensioniert.

Drehencoder

Der Drehencoder ist ein drehbarer Schalter, aus dem sich die Drehrichtung auslesen lässt. Daneben enthält der hier verwendete Drehencoder (PEC-12) eine Klickfunktion, d. h. man kann vielfältige Bedienoptionen mit nur einem Bedienelement realisieren. Umgesetzt wird die Klickfunktion durch einen normalen Tastschalter, der zwei Pins besitzt. Die Drehfunktion wird intern mit zwei Schaltern realisiert, anhand deren Schaltfolge die Drehrichtung erkannt werden kann. Diese zwei Schalter besitzen insgesamt wiederum vier Pins, wobei ein Pin (GND) von beiden Schaltern gemeinsam verwendet wird, sodass nur drei Pins nach außen geführt sind. Insgesamt besitzt der Drehencoder fünf Pins [6, 49].

Mit dem Drehencoder steuert der Fahrer die Smartphone App während der Fahrt.

6.2. Schaltplan

Auf Abbildung 6.1 ist ein schematischer Aufbau aller Hardwarekomponenten dargestellt.

Auf dem Steckbrett in der unteren Hälfte des Bildes sind von links nach rechts, zunächst die Bedienelemente zu sehen. Der Drehencoder, daneben die vier Taster, wobei der Taster am weitesten links die Bremse repräsentiert, die an einem Interrupt Port angeschlossen wird. Neben den Schaltern sind die zwei analogen Sensoren, der Temperatur- und der Stromsensor zu sehen.

In der oberen rechten Hälfte des Bildes befinden sich die Motorsteuerung (mit der Beschriftung ELMO), die über den MAX232 mit einer seriellen Schnittstelle des Arduino

Controllers verbunden ist. Weiterhin sind an der Motorsteuerung der Motor und die Stromversorgung vom Batteriemanagement (BMS) und daran wiederum die Akkus für den Antrieb angeschlossen.

Nur angedeutet ist die Verbindung zum Smartphone, dargestellt durch den Androiden. Aus Gründen der Übersichtlichkeit fehlt die 12 V Spannungsversorgung, die den Controller, Motorsteuerung und Batteriemanagement versorgt. Ebenso ist der Stromsensor in Wirklichkeit mit dem Motor verbunden, um dessen Stromverbrauch zu messen. Auch befindet sich der Temperatursensor in unmittelbarer Nähe zum Motor, um dessen Temperatur korrekt zu messen.

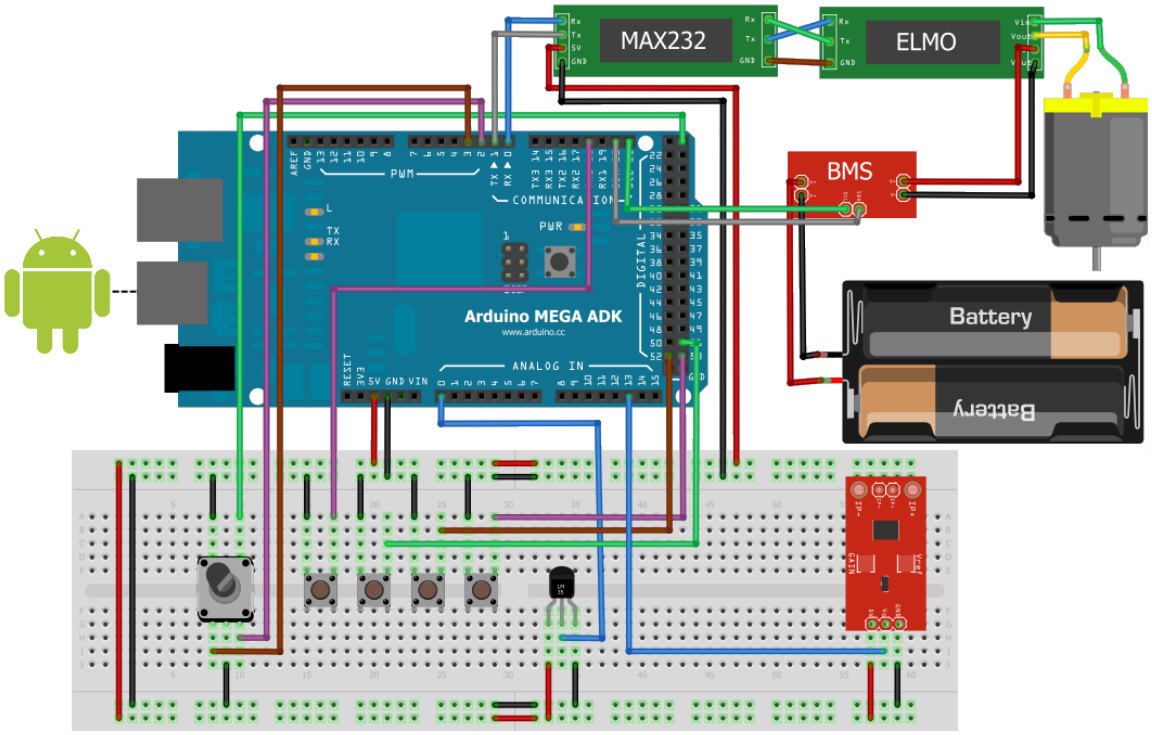


Abbildung 6.1.: Schaltplan

KAPITEL

SIEBEN

SOFTWARE

7.1. Entwicklungsumgebung

Da sich die Arduino Boards besonders an Einsteiger richten, sind Framework und Editor sehr einfach aufgebaut. Für die Realisierung dieses Projekts wurde auf die weit umfangreichere Entwicklungsumgebung Eclipse (Version 3.7) zurückgegriffen. Dazu existiert eine Anleitung, in der erklärt wird, wie das Arduino SDK in Eclipse integriert wird, damit der Code in Eclipse kompiliert und auf das Board geladen werden kann [3, 4, 11].

7.2. Verwendete Bibliotheken

Neben dem Arduino SDK wurden weitere Bibliotheken verwendet:

- **Arduino Accessory Bibliothek:** verbindet Mobiltelefon mit Controller per USB [16].
- **Bounce Bibliothek:** kann Eingabedaten von Schaltern verbessern, indem sie entprellt werden [2]. Entprellung verhindert ein instablies Verhalten des Schaltersingals beim Drücken bzw. Loslassen des Schalters [35, S. 51f].
- **ByteBuffer Bibliothek:** stellt eine Queue zur Verfügung [5].

7.3. Aufbau der Software

Überblick Arduino Software

Ein Arduino Programm muss stets eine `setup()` Funktion besitzen, die genau ein Mal beim Start aufgerufen wird. In diesem Funktionsaufruf werden Initialisierungsaufgaben erledigt, wie z. B. bei hier realisierten Anwendung: Anlegen der ByteBuffer, Setzen der Eingabepins, Zuweisung aufzurufender Funktionen zu den einzelnen Interruptpins.

Daneben gibt es noch eine `loop()` Funktion, die ständig durchlaufen wird und sich wiederholende Prozesse erledigt. Darunter fallen z. B., das Auslesen der Schalter und Sensoren, das Senden von Befehlen zur Motorsteuerung und die Kommunikation mit dem Mobiltelefon [35, S. 13ff].

Überblick Bedienelemente

Start	
Funktion	Dieser Schalter dient zum Start des Motors, ein erneuter Tastendruck stoppt den Motor.
Anbindung	Er wird über die `loop()` Funktion ausgelesen.
A+	
Funktion	Dieser Schalter dient dazu den Strom, mit dem der Motor aktuell versorgt wird, um 0,1 A zu erhöhen. Wird diese Taste gedrückt gehalten, wird der Strom solange um 0,1 A erhöht bis er losgelassen wird oder der Maximalwert von 10 A erreicht ist.
Anbindung	Er wird über die `loop()` Funktion ausgelesen.
A-	
Funktion	Dieser Schalter dient dazu den Strom, mit dem der Motor aktuell versorgt wird, um 0,1 A zu verringern. Wird diese Taste gedrückt gehalten, wird der Strom solange um 0,1 A verringert bis er losgelassen wird oder der Minimalwert von 0 A erreicht ist
Anbindung	Er wird über die `loop()` Funktion ausgelesen.
Bremse	
Funktion	Dieser Schalter ist beim Fahrzeug als binärer Drucksensor im Bremshebel integriert, der aber genau wie ein Schalter funktioniert, weshalb er in Abbildung 6.1 auch als Schalter dargestellt ist. Die Betätigung dieses Schalters führt dazu, dass sich der Motor abschaltet. Der Bremsvorgang selbst findet dann durch ein mechanisches Bremssystem statt.
Anbindung	Da es sich beim Bremsvorgang um einen sicherheitskritischen Prozess handelt, der möglichst schnell verarbeitet werden muss, wird die Bremse an einen Interrupt Port angeschlossen und außerhalb der `loop()` Funktion behandelt.
Drehencoder	
Funktion	Die Dreh- und Klickfunktion des Encoders werden zur Steuerung der Android App verwendet und in Kapitel IV genauer beschrieben.
Anbindung	Die Klickfunktion des Encoders wird in der `loop()` Funktion behandelt. Die Drehfunktion wird mit Hilfe eines in einem Xilinx Handbuch beschriebenen Algorithmus ausgelesen. Da sich beim Testen des Schalter herausgestellt hat, dass die Drehfunktion an Interrupt Ports besser erkannt wird, sind die zwei Pins für die Drehfunktion ebenfalls an Interrupt Pins angeschlossen [49].

7.4. Kommunikation Controller - Smartphone

In diesem Abschnitt wird der Ablauf der Kommunikation zwischen dem Android Smartphone und dem Controller näher erläutert, dabei wird zunächst das allgemeine Android Accessory Protokoll vorgestellt, anschließend das Datenformat, das für diese Arbeit verwendet wurde.

Dabei ist zu beachten, dass die Kommunikation nur in eine Richtung und zwar vom Controller zum Mobiltelefon stattfindet. Somit ist der Controller für keine Funktion (außer der Anzeige der Fahrdaten) auf das Smartphone angewiesen, womit das Fahrzeug auch ohne Smartphone gesteuert werden kann.

Android Accessory Protokoll

Zur Übertragung wird das Android Accessory Protokoll benutzt, das von Google selbst für Arduino und Smartphone verfügbar ist und auf dieser Seite im Detail vorgestellt wird [16]. Hier folgt nun eine Darstellung des Kommunikationsaufbaus:

1. Warten auf angeschlossene Geräte, indem laufend überprüft wird (z. B. in `loop()`) ob ein neues Gerät angeschlossen wurde und ob dieses den Accessory Modus unterstützt.
2. Ermittlung, ob angeschlossene Geräte den Accessory Modus unterstützen, indem die Vendor und Product ID des Gerätes überprüft werden. Sollte die Vendor ID mit Googles eigener ID (0x18D1) übereinstimmen und ebenso die Produkt ID eine der zwei festgelegten Werte (0x2D00 bzw. 0x2D01) haben, befindet sich das angeschlossene Gerät bereits im Accessory Modus, d. h. die Verbindung kann aufgebaut werden (siehe Schritt 4). Falls nicht muss in Schritt 3 versucht werden, das Gerät in den Accessory Modus zu bringen.
3. Versuch angeschlossene Geräte im Accessory Modus zu starten, wird durchgeführt, wenn Product oder Vendor ID nicht mit den Vorgaben (siehe Schritt 2) übereinstimmen. Dazu wird zunächst eine „Get Protocol“ Anfrage an das angeschlossene Gerät gestellt, mit der die unterstützte Protokollversion des Android Accessory Protokolls (bis jetzt existiert nur Version 1) abgefragt wird. Falls das angeschlossene Gerät mit der entsprechenden Protokollversion antwortet, werden die Anmeldeinformationen (siehe Code im Folgeabschnitt) an das Gerät geschickt. Anschließend sollte das Smartphone sich mit den in Schritt zwei erwähnten Vendor und Produkt IDs neu am Controller anmelden und Schritt eins und zwei sollten erfolgreich ablaufen können.
4. Verbindungsaufbau, falls Accessory Protokoll erfolgreich gestartet wurde. Die Endpunkte der Kommunikation teilt das Smartphone dem Controller über die Produkt ID mit, mit der es sich neu beim Accessory angemeldet hat, da für beide IDs jeweils festgelegte Schnittstellen existieren.

Umsetzung auf dem Arduino

Folgendes Codebeispiel illustriert, wie das Protokoll auf dem Arduino eingesetzt wird.

```
1  AndroidAccessory acc("TUfastEco",        // Hersteller
2      "Headunit",                          // Modell
3      "Headunit TUfast Eco",               // Beschreibung
4      "1.0",                               // Version
5      "ecocontrolpaneltest.appspot.com",   // URL
6      "0815");                             // Seriennummer
7
8  uint8_t msg[46];                         // Daten zum Versand
9
10 void loop() {
11
12   if (acc.isConnected()) {
13     acc.write(msg, 46);                  // Versand der Nachricht
14   }
15
16 }
```

Das acc Object enthält erhält bei seiner Instanzierung die bereits in vorhergehenden Abschnitt erwähnten Anmeldedaten. In der `loop()` Funktion wird über den Methodenaufruf `isConnected()` bei jedem Aufruf (siehe Schritt 1 vorhergehender Abschnitt) der Verbindungsstatus überprüft und ggf. versucht eine Verbindung aufzubauen. Im Erfolgsfall versendet die Methode `acc.write()` das Array msg, das die gewünschten Daten enthält.

Eingesetzte Datenformate

Nun werden die unterschiedlichen Nachrichtentypen vorgestellt, d. h. welche Daten im msg Array des vorhergehenden Abschnitts verschickt werden. Es existieren drei verschiedene Nachrichtentypen (Sensordaten, Befehle zur Motorsteuerung, Antworten der Motorsteuerung), die im Anhang unter A.1 im Detail dargestellt sind.

Solange keine Kommunikation mit der Motorsteuerung stattfindet, werden nur Sensordatenpakete verschickt. Sollten Befehle bzw. Antworten der Motorsteuerung vorliegen, wird dennoch gewährleistet, dass mindestens jedes dritte Datenpaket Sensordaten enthält, sodass die Anzeige aktueller Fahrzeugdaten möglichst wenig durch die Übertragung von Debugdaten verzögert wird.

Teil IV.

Smartphone

KAPITEL

ACHT

BENUTZEROBERFLÄCHEN & BEDIENUNG

In diesem Kapitel wird die Android App mit ihren Funktionen und verschiedenen Oberflächen vorgestellt.

8.1. Instrumententafel

Abbildung 8.1.: Instrumententafelansicht

Abbildung 8.1 zeigt die App, wie sie sich dem Fahrer während der Fahrt präsentiert.

Die zwei Balken auf der linken Seite dieser Ansicht zeigen die Temperatur des Sensors in Grad Celsius und den Ladezustand der Batterie in Prozent an.

In der Mitte der Instrumententafel ist am oberen Rand ein Timer zu sehen, der beim

Starten des Fahrzeugs ebenfalls gestartet wird. Darunter auf der linken Seite ist ein Tachometer zu sehen, an dem die aktuelle Geschwindigkeit in km/h abzulesen ist. Darunter ist die Geschwindigkeit und darunter der aktuelle Kilometerstand als digitale Zahl dargestellt. Der Tachometer auf der rechten Seite zeigt die aktuelle Leistung des Motors an. Darunter ist wiederum die Leistung als Zahl und weiter unterhalb die bisher geleistete Arbeit des Motors dargestellt. Unterhalb der digitalen Anzeigen ist der Rundenzähler zu sehen, der die aktuelle Runde des Rennens anzeigt.

Auf der rechten Seite der Instrumententafel sind Spannungs- und Stromwerte dargestellt, mit denen der Motor aktuell versorgt wird.

Unterhalb der Instrumente sind links die Bedienelemente zu sehen, die ein Benutzer am Touchscreen anklicken kann. Diese Schaltflächen sind aber auch mit Hilfe des Drehencoders am Lenkrad des Fahrzeugs bedienbar. Durch Drehen am Encoder kann ein Bedienelement fokussiert werden, in Abbildung 8.1 ist beispielsweise aktuell der Call Button fokussiert. Durch Klicken kann jetzt ein Anruf getätigt werden. Wird die Volume Schaltfläche angeklickt, behält diese Taste den Fokus und mittels Drehfunktion kann die Lautstärke des aktuellen Anrufs in mehreren Stufen erhöht und erniedrigt werden. Bei erneutem Klicken können wieder andere Schaltflächen selektiert werden. Die letzte Taste mit der Beschriftung Reset kann Timer, Kilometerstand und den Zähler für geleistete Arbeit wieder zurücksetzen. Um zufälliges Anklicken zu vermeiden, muss der Reset mit zwei Klicks bestätigt werden.

Neben den Schaltflächen finden sich noch drei kleinere Symbole, die den Status der Kommunikationskanäle durch die Farben grau, rot und grün anzeigen. Eine Schaltfläche ist grau (wie zum Beispiel NET und GPS auf Abbildung 8.1), solange der Kommunikationskanal nicht verwendet wurde. Wird der Kommunikationskanal erfolgreich verwendet ändert sich die Farbe auf grün. Jeder Button besitzt einen eigenen Timeout, der bei Ablauf die Farbe auf rot ändert (wie z. B. USB in der Abbildung). Diese Ansicht dient dazu, mögliche Probleme schnell eingrenzen zu können.

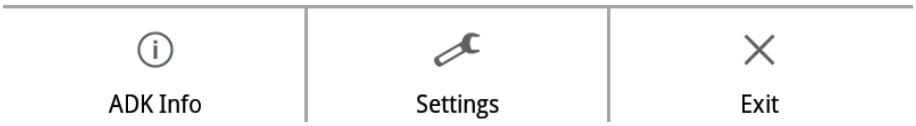


Abbildung 8.2.: Optionsmenü

Beim Drücken der Optionstaste erscheint der in Abbildung 8.2 dargestellte Dialog. Dieser Dialog dient zum Starten der weiteren Oberflächen bzw. zum Beenden der Anwendung. Dieser Dialog kann allerdings nicht mehr mit den Schaltern am Lenkrad bedient werden, da diese Ansichten während der Fahrt nicht benötigt werden.

8.2. Einstellungen

Der Einstellungsdialog ist auf Abbildung 8.3 zu sehen.

In dieser Ansicht kann die Übertragung zum Server und die Adresse des Webservers,

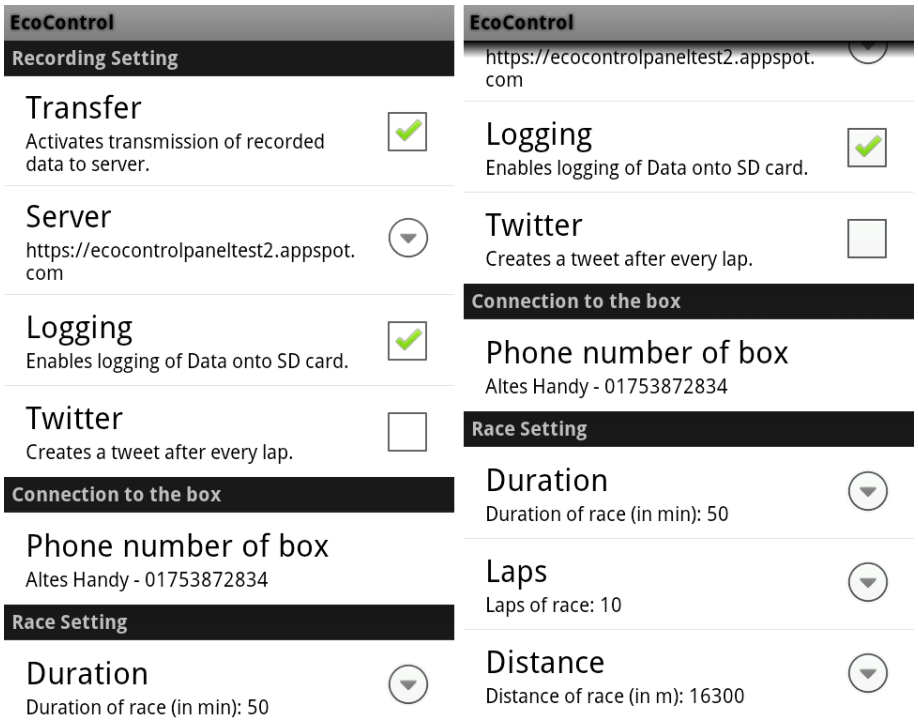


Abbildung 8.3.: vollständige Einstellungsansicht

der die Messdaten archiviert, eingestellt werden. Ferner können die Daten zusätzlich auf dem internen Speicher des Smartphones gesichert werden. Ist die Twitter Option aktiviert, wird eine kurze Nachricht auf dem Twitterkanal des TUfast Eco-Teams veröffentlicht, wenn eine Runde während des Rennens erfolgreich absolviert wurde.

Auch die Nummer der Box, die von der Call Schaltfläche in der Instrumententafel gewählt wird, kann aus dem im Telefon gespeicherten Nummern ausgewählt werden.

Zuletzt können die Parameter zum Rennen selbst eingestellt werden. Darunter fallen die Dauer, die Länge der Rennstrecke und die Rundenanzahl.

8.3. Debugoberfläche

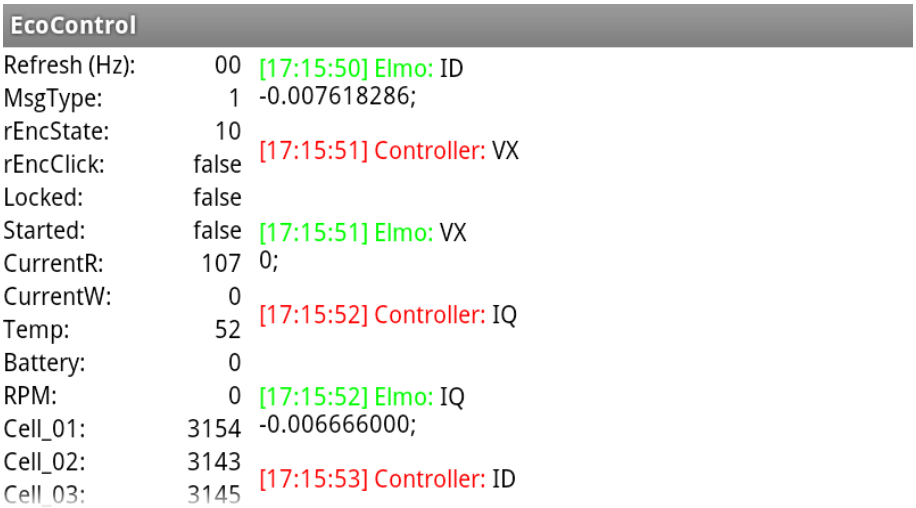


Abbildung 8.4.: Debugansicht

Die Debugoberfläche ist auf Abbildung 8.4 zu sehen. Sie dient zur Darstellung der Rohdaten, die vom Controller zum Smartphone gesendet werden. Dazu werden links die Nachrichten, die die Sensordaten enthalten, in Tabellenform dargestellt. In der rechten Hälfte sind die Nachrichten, die vom Controller zur Motorsteuerung gesendet wurden und deren Antworten zu sehen.

KAPITEL

NEUN

SOFTWARE

9.1. Entwicklungsumgebung

Für die Implementierung der App wurde ebenfalls Eclipse (Version 3.7) eingesetzt, in Verbindung mit dem von Google selbst bereitgestellten Plugin, den sog. „Android Developer Tools“ für die Entwicklung von Android Apps [11, 14].

9.2. Verwendete Frameworks & Bibliotheken

Hier werden die wichtigsten Bibliotheken aufgelistet, die zusätzlich zum Android Framework (in Version 10) verwendet werden.

- **Guice 3.0 + Roboguice 2.0** sind Frameworks, die es ermöglichen Objekte per Dependency Injection zu erzeugen und zu verwenden [20, 48]. Verwendet werden sie, da sie das Programmieren erleichtern. Indem sie automatisch Klassen instanzieren, verringern sie die Menge an Code und vereinfachen die Erzeugung komplexer Objekte mit vielen Abhängigkeiten. Weitere Informationen zu diesen Frameworks finden sich in Abschnitt 9.3.
- **JKalman** enthält eine Implementierung eines Kalman Filters, der für die Verarbeitung der Sensorsignale eingesetzt wird [43]. Diese Bibliothek erlaubt eine einfache Handhabung dieses mathematisch anspruchsvollen Filters. In der hier beschriebenen App wird dieser Filter für die Glättung der Positionssignale eingesetzt, da beim Test mit dem Smartphone Abweichungen von bis zu 10 m zwischen zwei Messpunkten festgestellt wurden. Diese Abweichungen verschwinden zwar auch durch den Filter nicht, aber der gefilterte Kurs entspricht eher dem gefahrenen Kurs als der ungefilterte.
- **Jackson 1.8.5** ist ein Parser, der verwendet wird, um Daten in die JavaScript Object Notation (JSON) zu konvertieren, die an den Webservice übermittelt werden [42].

Eingesetzt wird dieser Parser, da sich mit ihm sehr performant und einfach JSON Objekte erzeugen lassen.

9.3. Architektur

MVC

MVC ist eine Architektur, die 1979 von Trygve Reenskaug entwickelt wurde. Hierbei wird die Anwendung in die namensgebenden Teile Model, View und Controller aufgeteilt [34].

Das **Model** im MVC beinhaltet die gesammelten Informationen. Im konkreten Fall (siehe Abbildung 9.1) existieren zwei SQLite Datenbanken. Die erste Datenbank enthält Sensordaten, wird von der MeasurementDataDbHelper Klasse verwaltet und dient als Zwischenspeicher bis die Daten vom Smartphone an den Webservice geschickt werden. Die zweite Datenbank enthält Befehle und Antworten der Motorsteuerung. Diese Datenbank wird von der DebugDataDbHelper Klasse verwaltet, deren Daten zur Fehlersuche bei der Kommunikation zwischen Controller und Motorsteuerung dienen. Die Schemata beider Datenbanken finden sich im Anhang B.1. Die jeweiligen Helper Klassen verwalten die Datenbanken, d. h. sie erzeugen, aktualisieren, öffnen oder schließen die Datenbanken. Der Zugriff auf die Datenbanken erfolgt nicht über die Helper Klassen sondern über die beiden Klassen DebugDataAccessObject und MeasurementDataAccessObject, die Methoden mit Abfragen für den einfachen Datenzugriff enthalten. Daneben enthalten die beiden AccessObject Klassen jeweils einen ContentObservable, an dem sich Views anmelden können, um über Aktualisierungen im Datenbestand informiert zu werden. Als Datenquelle dienen Klassen im datasource Paket, die die gesammelten Informationen in die Datenbanken schreiben. Daneben befinden sich noch zwei weitere Klassen im Model, zum einen die ButtonStateMachine, ein Zustandsautomat für den Drehencoder am Lenkhebel. Zum anderen die Klasse MeasurementData, die Berechnungen mit Hilfe der gesammelten Sensordaten durchführt, z. B. Berechnung der Geschwindigkeit. Die einzelnen Stufen der Datenverarbeitung werden im Kapitel Datenfluss näher vorgestellt.

Die Aufgabe von **Views** ist die Darstellung des Nutzerinterfaces. Beim Android Framework wird das Design mit Hilfe von XML Dateien beschrieben. Dieses Layout wird dann mit Hilfe von Activity Klassen geladen. Diese Activity Klassen kümmern sich allerdings nicht ausschließlich um die graphische Darstellung, sondern übernehmen auch Aufgaben, die normalerweise vom Controller übernommen werden. In Abbildung 9.1 werden diese Activity Klassen dem View Paket zugeordnet, es wäre aber durchaus möglich nur die XML Dateien zur View Kompomente zu zählen und die Activity Klassen dem Controller zuzuordnen. Die Abbildung zeigt ferner, dass jede der bereits vorgestellten graphischen Oberflächen ihre eigene Activity Klasse besitzt. Zusätzlich ist auch noch die ContactListActivity dargestellt, die für die Auswahl der Rufnummer der Box genutzt wird, aber im vorherigen Kapitel nicht explizit vorgestellt wurde. Die gestrichelten Linien zwischen den Activities sollen andeuten, wie die einzelnen Oberflächen miteinander verknüpft sind, d. h. von welcher Oberfläche die jeweils andere erreichbar ist. Der DbObserver, den ControllerDebugActivity und ControlActivity besitzen, ist mit der Datenbank verbunden und sorgt bei Bedarf für die Aktualisierung der dargestellten Informationen, näheres dazu im

Abschnitt über den Datenfluss.

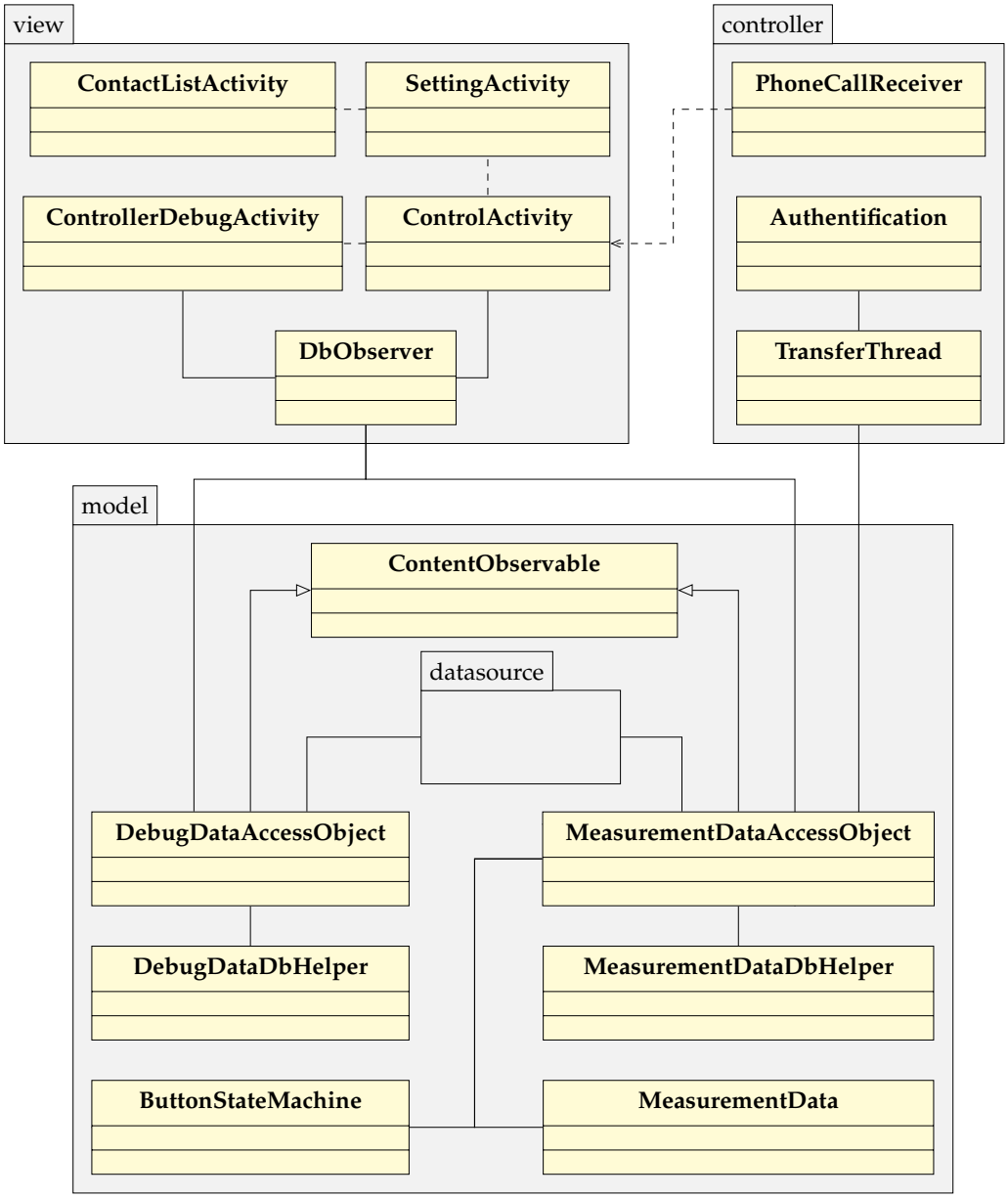


Abbildung 9.1.: Klassendiagramm - Architektur der Android App

Die Aufgabe eines **Controllers** ist laut Reenskaug die Verbindung von Nutzereingaben mit dem System. Im konkreten Fall befindet sich im Controller unter anderem der PhoneCallReciever, eine Klasse, die eingehende Anrufe abweisen kann, falls die ControlActivity gestartet ist. Daneben existiert der TransferThread, der mit Hilfe der Authentification Klasse die Anmeldung am Server übernimmt und mehrmals in der Minute versucht Daten zum Server zu übertragen. Die Nutzereingaben an die genannten Komponenten, erfolgt

allerdings nicht direkt, wie bei MVC üblich, sondern über Einstellungen, die vom Nutzer festgelegt wurden.

Dependency Injection

In der vorliegenden App werden die beiden Dependency Injection Frameworks Guice, und dessen Portierung auf das Android Framework Roboguice verwendet. Diese Frameworks kümmern sich um die automatische Instanzierung von Klassen.

Klassen, die von den Frameworks erzeugt werden sollen, erhalten eine Annotation. Die Annotation `@Singleton` bewirkt zum Beispiel, dass eine so gekennzeichnete Klasse nur genau ein einziges Mal innerhalb des Programms instanziert wird. Klassen, die dieses Objekt verwenden, erhalten eine Klassenvariable dieses Typs, die mit einer `@Inject` Annotation versehen wird. Diese Annotation bewirkt, dass das erzeugte AccessObject in diese Klasse „injiziert“ wird.

Folgender Codeauszug wird auch in der App verwendet:

```
1  //Erzeugung genau eines Objekts
2  @Singleton
3  public class MeasurementDataAccessObject {
4    //...
5  }
6
7  //Verwendung von MeasurementDataAccessObject
8  public class ControlActivity extends RoboActivity{
9    //Injizierung des Objekts
10   private @Inject MeasurementDataAccessObject mao;
11 }
```

Diese Einbindung von Objekten mit Hilfe der `@Inject` Annotation erspart dem Programmierer, einerseits die Initialisierung mit Hilfe von `new` und ermöglicht so kürzeren und leichter lesbaren Code. Andererseits stellt das Framework aber auch automatisch sicher, dass nur genau eine Instanz von MeasurementDataAccessObject existiert. Ein weiterer Vorteil ist, dass eine Injizierung mittels `@Inject` nicht nur in einer Klasse, sondern auch in mehreren Klassen stattfinden kann. Was sich gerade für die verwendeten Datenbanken in diesem Projekt anbietet, da diese von mehreren Klassen benötigt werden, wie man in Abbildung 9.1 erkennen kann.

Eine Besonderheit von Roboguice ist die vereinfachte Erzeugung von Objekten aus den XML Layout Dateien von Android. Folgendes Beispiel zeigt die Einbindung einer TextView mit der ID speed. Im ersten Fall ohne Dependency Injection muss für das TextView Objekt eine Klassenvariable angelegt werden, die dann beim Start der Activity (d. h. in der onCreate() Methode) mit Hilfe von findViewById() initialisiert wird. Mit Dependency Injection genügt es die Definition der Klassenvariable mit `InjectView(R.id)` gefolgt von der id der gewünschten View zu annotieren. Somit wird pro View eine Zeile Programmiercode gespart, was gerade bei den vielen Views z. B. in der ControlActivity ein beachtliche Menge Code einspart. Dabei ist zu beachten, dass ControlActivity anstelle der Standardklasse Activity von der Klasse RoboActivity der RoboGuice Bibliothek erben muss, um die

Dependency Injection verwenden zu können.

```
1  //Ohne Verwendung von Dependency Injection
2  public class ControlActivity extends Activity {
3    private TextView speedView;
4    public void onCreate(Bundle savedInstanceState) {
5      speedView = (TextView) findViewById(R.id.speed);
6    }
7  }
8
9  //Mit Verwendung von Dependency Injection
10 public class ControlActivity extends RoboActivity {
11   //Injizierung des Objekts
12   private @InjectView(R.id.speed) TextView speedView;
13 }
```

9.4. Datenfluss

Die folgenden Sequenzdiagramme zeigen den Fluss der Daten durch das Programm. Der Übersicht halber wurden mehrere Diagramme verwendet und ebenso die Parameter der Methodenaufrufe weggelassen.

Abbildung 9.2 zeigt den Weg der Daten vom Empfang am Smartphone durch den AccessoryReaderThread (als Teil des datasource Pakets auf Abbildung 9.1) in die Datenbank (MeasurementDataAccessObject) mit den zwei Hilfsklassen MeasurementData und ButtonStateMachine.

Wie in Abbildung 9.1 dargestellt, erben die Datenbanken DebugDataAccessObject und MeasurementDataAccessObject von ContentObservable, d. h. Subklassen von ContentObserver, also im konkreten Fall Klassen, die den DbObserver verwenden, können sich bei den Datenbanken registrieren und werden über Updates informiert. Diese Aktualisierungsbenachrichtigungen werden im notify() Aufruf in Abbildung 9.2 verschickt. Danach können sich die registrierten Observer, also z. B. ControlActivity die gewünschten Daten vom MeasurementDataAccessObjekt holen. Dieser Vorgang wird in Abbildung 9.3 dargestellt, da die Aktualisierung des Nutzerinterfaces nicht mehr im AccessoryReaderThread geschieht, sondern in einem eigenen Thread, d. h. ein Update kann zu einem beliebigen Zeitpunkt erfolgen.

Die Verarbeitung der Befehle und Antworten der Motorsteuerung wird durch die Diagramme nicht abgedeckt, verläuft aber analog dazu. Dabei werden die Hilfsklassen MeasurementData und ButtonStateMachine nicht benötigt. Ebenso müssen MeasurementDataAccessObject durch DebugDataAccessObject und ControlActivity durch DebugActivity ersetzt werden.

Abbildung 9.4 zeigt links den PositionListener, der die LocationListener Schnittstelle des Android Frameworks implementiert und damit regelmäßig neue Positionsinformationen mittels eines onLocationChanged() Aufrufs erhält. Die GPS Daten werden mittels KalmanFilter gefiltert und anschließend werden die GPS Daten zusammen mit den aktuellsten Sensormesswerten vom Controller in die Datenbank (MeasurementDataAccessObject) eingetragen.

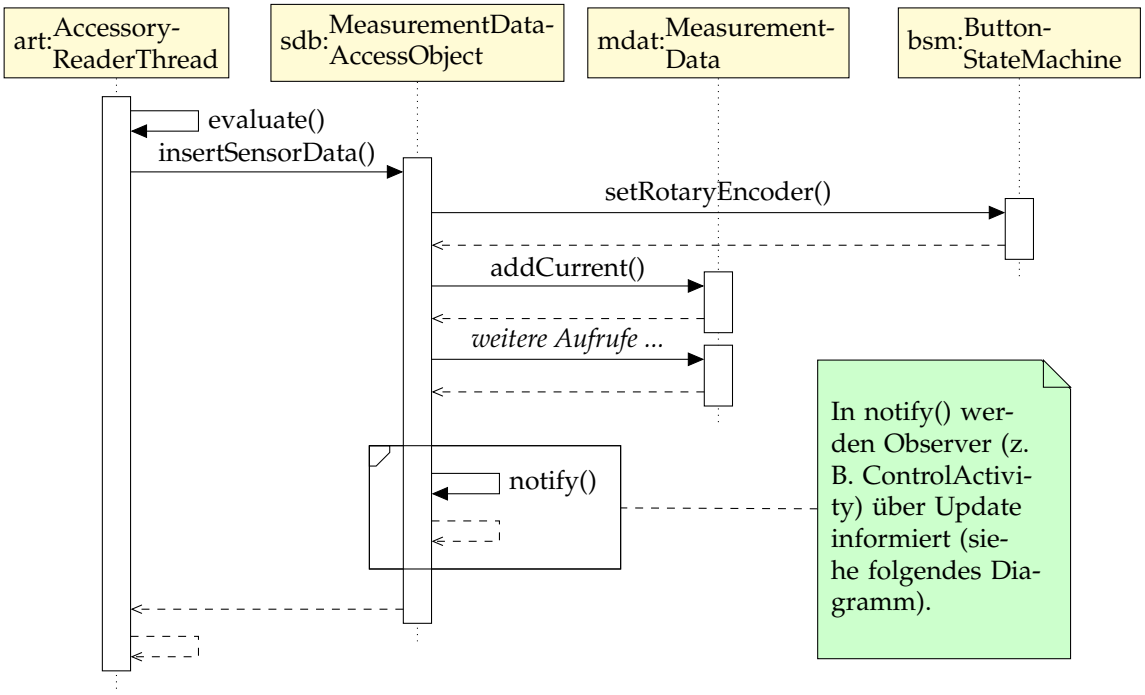


Abbildung 9.2.: Sequenzdiagramm - Datenfluss vom Empfang am Smartphone zur Datenbank

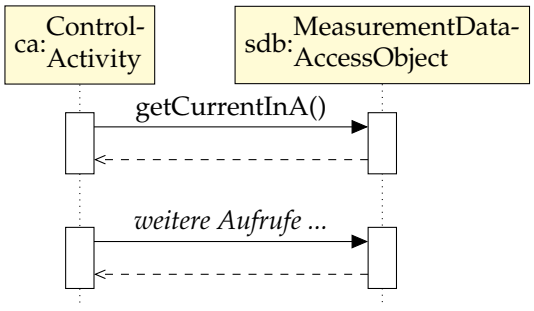


Abbildung 9.3.: Sequenzdiagramm - Aktualisierung der Nutzeroberfläche

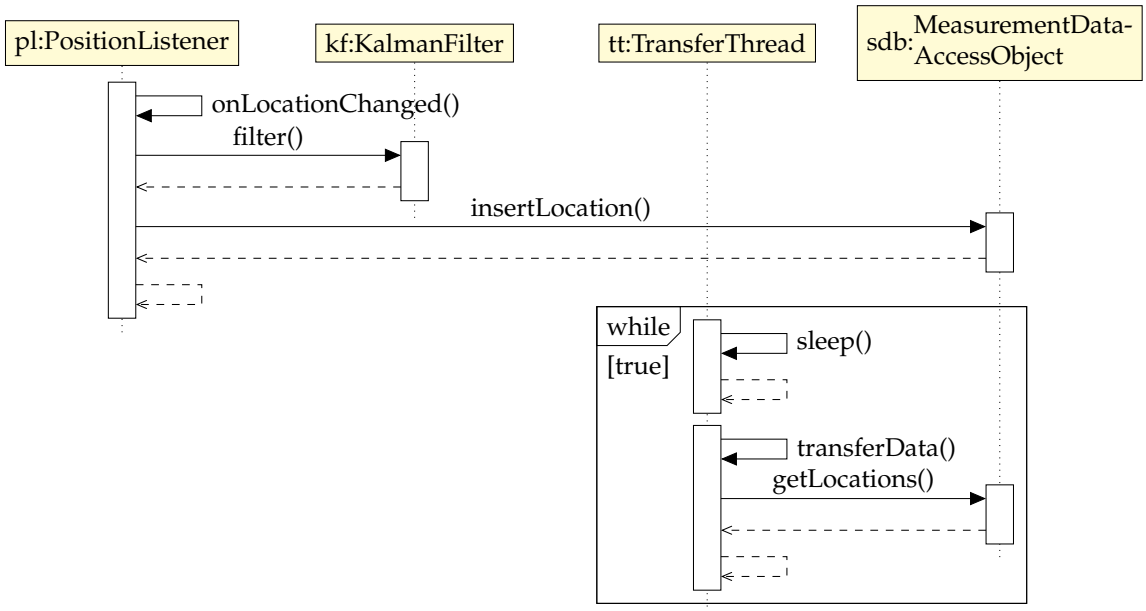


Abbildung 9.4.: Sequenzdiagramm - Sammeln von GPS Daten & Transfer zum Webservice

TransferThread versucht alle fünf Sekunden Daten zum Server zu übertragen. Dazu werden die Daten im transferData() Aufruf aus der Datenbank ausgelesen und zum Webservice übertragen. Im Erfolgsfall werden die Daten aus der Datenbank gelöscht, andernfalls verbleiben die Daten in der Datenbank und der TransferThread versucht beim nächsten Aufruf die Daten zu übertragen.

9.5. Verwendete Protokolle

Die Datenübertragung zum Server geschieht mittels Representational State Transfer, kurz REST. Die Bezeichnung REST stammt von Roy Fielding aus dem Jahr 2000 und steht für eine Art von Client Server Architektur, in der etablierte Techniken wie zum Beispiel HTTP, URLs und XML eingesetzt werden, um Dienste anzubieten [13].

Die hier vorgestellte Anwendung akzeptiert zum Beispiel die Sensordaten unter der URL *https://ecocontrolpanel.appspot.com/rest/accept*. Als Übertragungsprotokoll wird wie in der URL zu erkennen ist, HTTPS verwendet. Die Sensordaten selbst werden in einem POST Paket des HTTPS verschickt. Dabei werden die Daten in die JavaScript Object Notation (JSON) übersetzt. JSON ist ein Dateiaustauschformat ähnlich zu XML, das JavaScript Syntax nutzt um Daten zu formatieren [8].

Ein Beispiel für einen JSON formatierten Datensatz wie es von der Anwendung verwendet wird findet sich im Anhang unter A.2.

Die Wahl der Architektur fiel auf REST, da es zustandslos ist, d. h. es sind keine komplexen Zustände nötig, um Daten übertragen zu können. Im konkreten Fall wird zwar aus Sicherheitsgründen ein Anmelden am Webservice verlangt, aber diese Anmeldung findet

nur selten statt, da sich der Client über einen Cookie identifizieren kann und eine Neuanmeldung erst nach Ablauf des Cookies erforderlich wird. Die Zustandslosigkeit bietet sich besonders bei der möglicherweise instabilen Übertragung über das Mobilfunknetz an, da so bei einem Wiederverbinden, nicht erst langwierig ein Zustand wiederhergestellt werden muss, um Daten übertragen zu können.

Als Übertragungsformat wurde JSON ausgewählt, da es wie XML für Menschen leicht zu lesen ist, gut erweitert, aber im Gegensatz zu XML etwas kompakter geschrieben werden kann und damit auch weniger Daten zu übertragen sind.

9.6. Realisierung der Sprachkommunikation

Die vorhergehenden Abschnitte befassen sich größtenteils mit der Übermittlung der Sensordaten. Die Sprachkommunikation wird nun in diesem Abschnitt erläutert.

Für die Sprachkommunikation bietet sich SIP an, ein Voice over IP (VoIP) Protokoll, das seit Version 2.3 von Android selbst unterstützt wird. Damit ist diese Funktion einfach zu realisieren und für eine dauerhafte Verbindung kostengünstiger als die herkömmliche Telefonfunktion. Eine Kommunikationsfunktion mittels SIP wurde testweise in die App eingebaut, es stellte sich aber heraus, dass diese Verbindung, sogar bei guter Datenverbindung, mit hoher Latenz (mehrere hundert Millisekunden) und Paketverlusten (teilweise über 90 % Verlust) verbunden war, und damit nicht einsetzbar ist [15].

Der nächste Schritt war das Nutzen der Telefonfunktion, was ebenfalls in Android einfach möglich ist, indem ein Intent erzeugt wird, der einen Anruf startet. So wurde erfolgreich eine Verbindung zu einer angegebenen Nummer aufgebaut, allerdings wird hierbei die Telefonanwendung von Android gestartet. Die Telefonapp überdeckte bei ihrem Start die Instrumententafel. Die Telefonapplikation lässt sich zwar ohne Beenden der Sprachverbindung minimieren, aber diese Aktion muss an einer Taste am Telefon durchgeführt werden, was für den Fahrer während des Rennens verboten ist. Damit ist auch diese Lösung nicht praktikabel [23].

Realisiert wurde deshalb eine eigene Lösung, die allerdings Änderungen an der Telefonapplikation nötig machte. Die neue Telefonanwendung unterstützt zwei zusätzliche Aktionen für Intents, SILENT_CALL, das einen Anruf startet ohne die aktuell aktive App zu pausieren und SILENT_ACCEPT_CALL, das einen eingehenden Anruf annimmt ohne die aktuell aktive App zu pausieren. Da die Telefonanwendung bei einem herkömmlichen System nicht ohne weiteres ausgetauscht werden kann, wurde Android mit der geänderten Telefonapp neu kompiliert.

Verwendet wurde eine modifizierte Version des Kernels, dabei handelt es sich um die CyanogenMod 7.2 für das Nexus S. Anleitungen für Kompilierung und Installation finden sich unter [9, 10].

Die jetzige Lösung hat den Nachteil, dass die Telefonfunktion auf Systemen ohne die geänderte Telefonapp nicht funktioniert. Prinzipiell stellt die Verwendung der Telefonfunktion ohne Zutun des Nutzers auch ein Sicherheitsrisiko dar, da diese Funktion von jeder App benutzt werden kann und so unbemerkt Kosten entstehen können. Da das

Smartphone allerdings nur im Fahrzeug Verwendung findet und kaum zusätzliche Applikationen installiert werden, wird dieses Risiko in Kauf genommen.

Wie in Abbildung 9.1 zu sehen ist, existiert eine Klasse PhoneCallReceiver. Die Funktion dieser Klasse ist es, Anrufe, die nicht von der voreingestellten Nummer kommen, während die App gestartet ist, abzuweisen. Diese Funktion wurde eingeführt, damit der Fahrer während des Rennens nicht durch unnötige Anrufe gestört wird. Dabei werden Funktionen genutzt, die normalerweise vom Framework nicht für die Programmierer von Apps freigegeben sind. Wie diese Einschränkungen zu umgehen sind, wird in [30] erklärt.

Teil V.

Webservice

KAPITEL

ZEHN

BENUTZEROBERFLÄCHEN & BEDIENUNG

In diesem Kapitel wird die Webseite vorgestellt, die die Sensordaten des Webservices darstellt. Diese Webseite ist nicht frei zugänglich, sondern erfordert eine Anmeldung, damit konkurrierende Teams keine allzu detaillierten Daten über die Leistungsfähigkeit und eventuelle Schwachstellen des Fahrzeugs erhalten. Für interessierte Besucher steht eine separate Webseite zur Verfügung, die nur einen Teil der Daten darstellt.

10.1. Übersicht

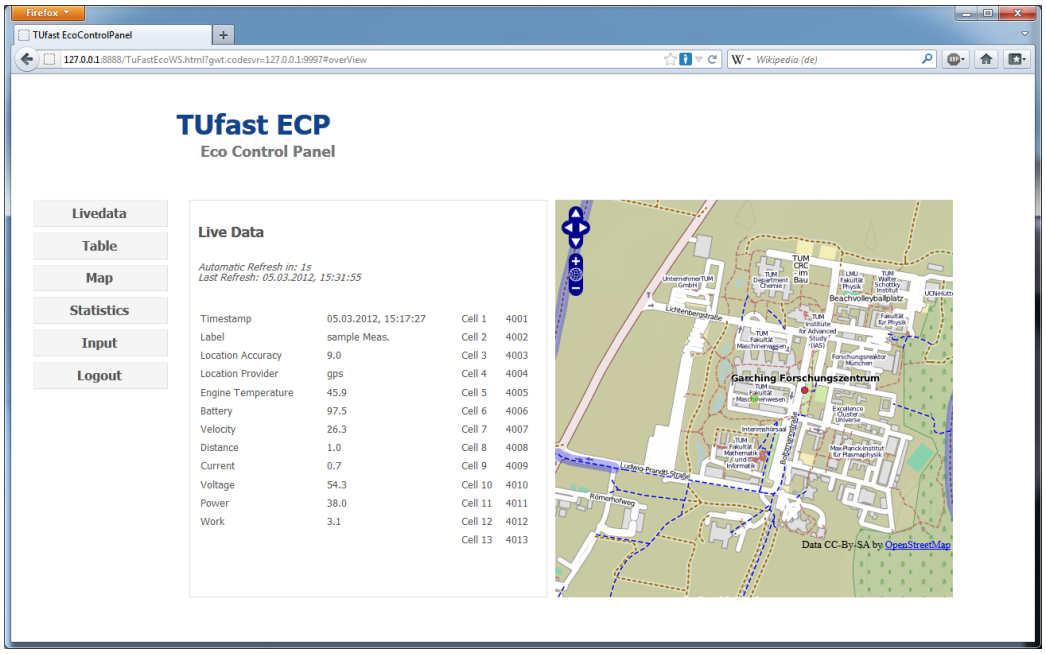


Abbildung 10.1.: Übersichtsseite

Abbildung 10.1 zeigt die Startseite der Webseite, die immer die aktuellsten Daten zum Fahrzeug anzeigt und sich dazu automatisch aktualisiert. Dargestellt sind auf der rechten Seite der Website eine kleine Landkarte, deren Mitte die letzte bekannte Position des Fahrzeugs zeigt. Daneben werden die aktuellen Daten wie z. B. Geschwindigkeit, Batterieladestand usw. tabellarisch dargestellt. Diese Ansicht soll den Mechanikern während des Rennens einen Überblick über den aktuellen Zustand des Fahrzeugs verschaffen.

10.2. Analyseoberflächen

Die Oberflächen in diesem Abschnitt dienen dazu, die gesammelten Sensordaten möglichst übersichtlich darzustellen, um die Daten einfach auswerten zu können.

Alle drei Oberflächen besitzen die Schaltflächen Previous, Next, Refresh und Delete. Next bzw. Previous dienen zur Navigation im Datenbestand, Refresh aktualisiert die dargestellten Daten und Delete schließlich löscht die aktuell angezeigten Daten.

Landkarte

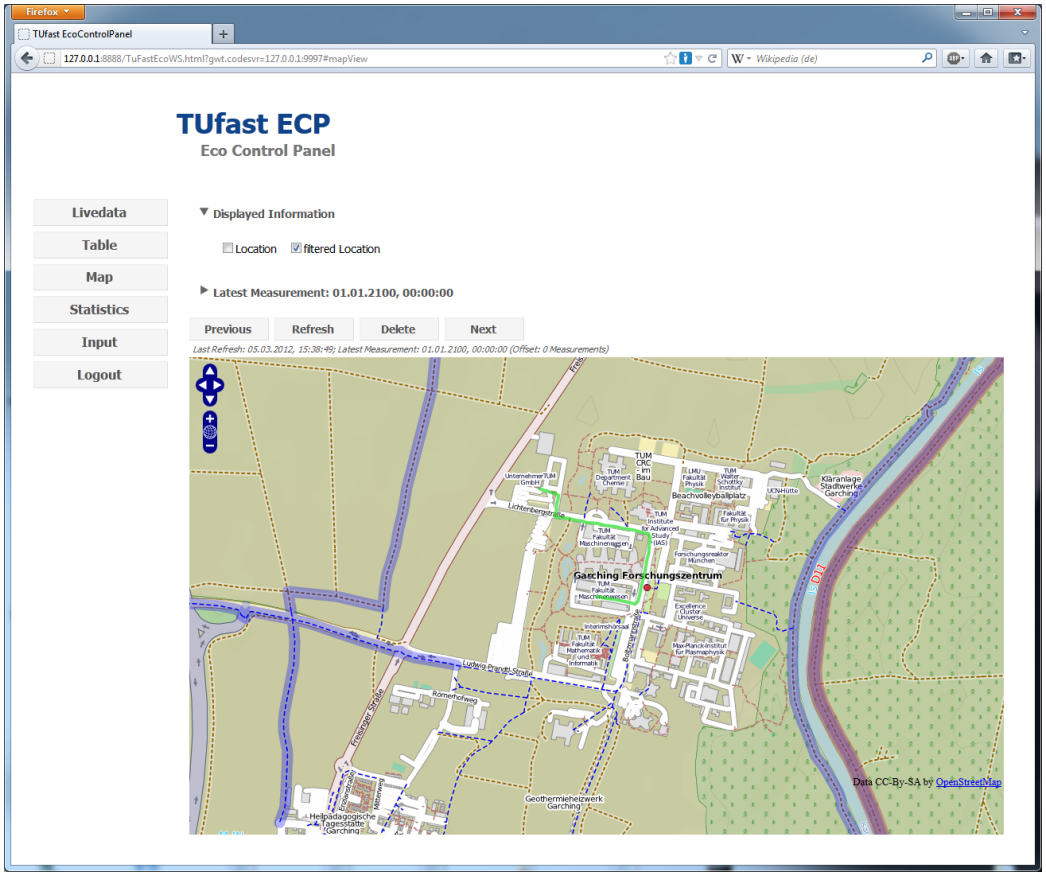


Abbildung 10.2.: Landkarte mit eingezeichnetem Kurs des Fahrzeugs

Abbildung 10.2 zeigt eine Landkarte, auf der der Kurs des Fahrzeugs als grüne Linie eingezeichnet wird. Als Einstellungsmöglichkeiten bei dieser Übersicht steht eine Auswahl zwischen gefilterten und ungefilterten Positionsdaten zur Verfügung. Ebenso kann ein Datum ausgewählt werden, sodass nur Daten berücksichtigt werden, die vor diesem Zeitpunkt entstanden sind.

Tabelle

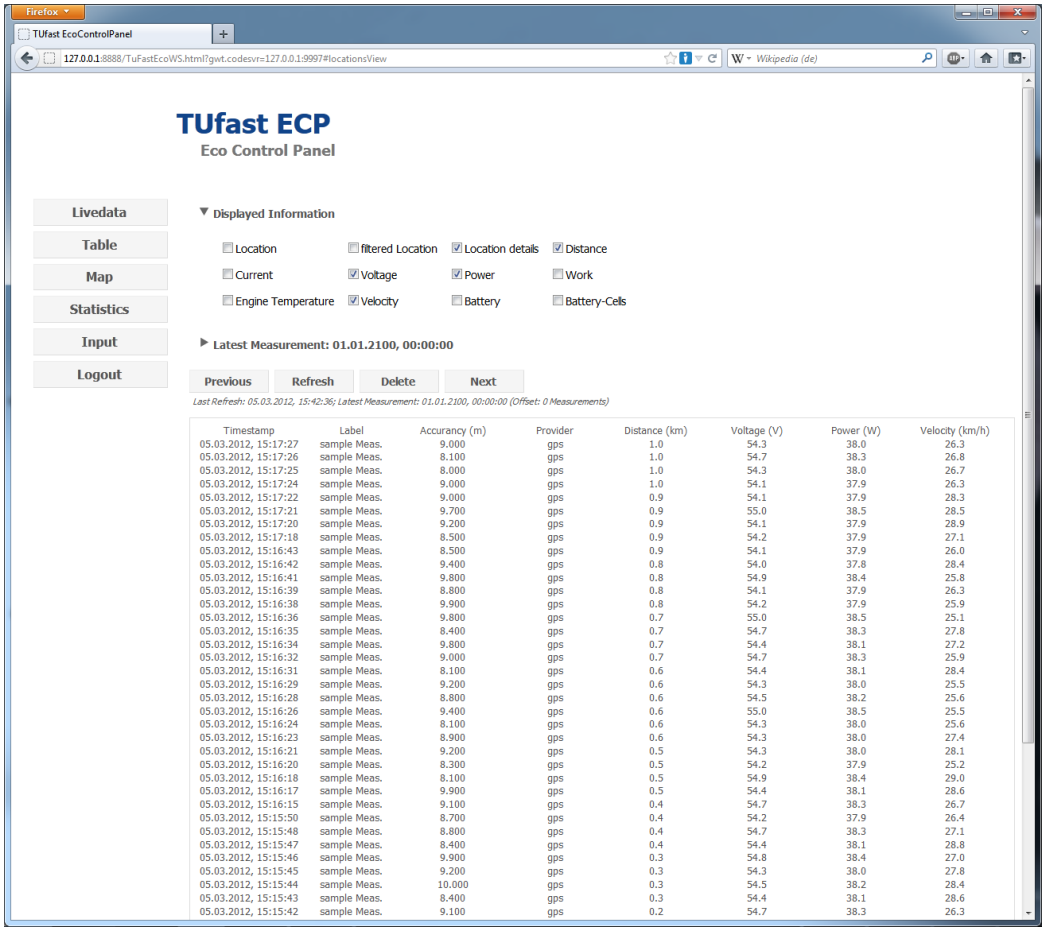


Abbildung 10.3.: Tabellarische Übersicht der Fahrzeugdaten

Abbildung 10.3 zeigt die tabellarische Übersicht. Dazu können Nutzer die gewünschten Daten auswählen, die die Tabelle darstellt. Diese Tabelle enthält die 50 letzten Werte der ausgewählten Daten. Zusätzlich kann ein Zeitpunkt gewählt werden, sodass nur Daten berücksichtigt werden, die vor diesem Datum aufgezeichnet wurden.

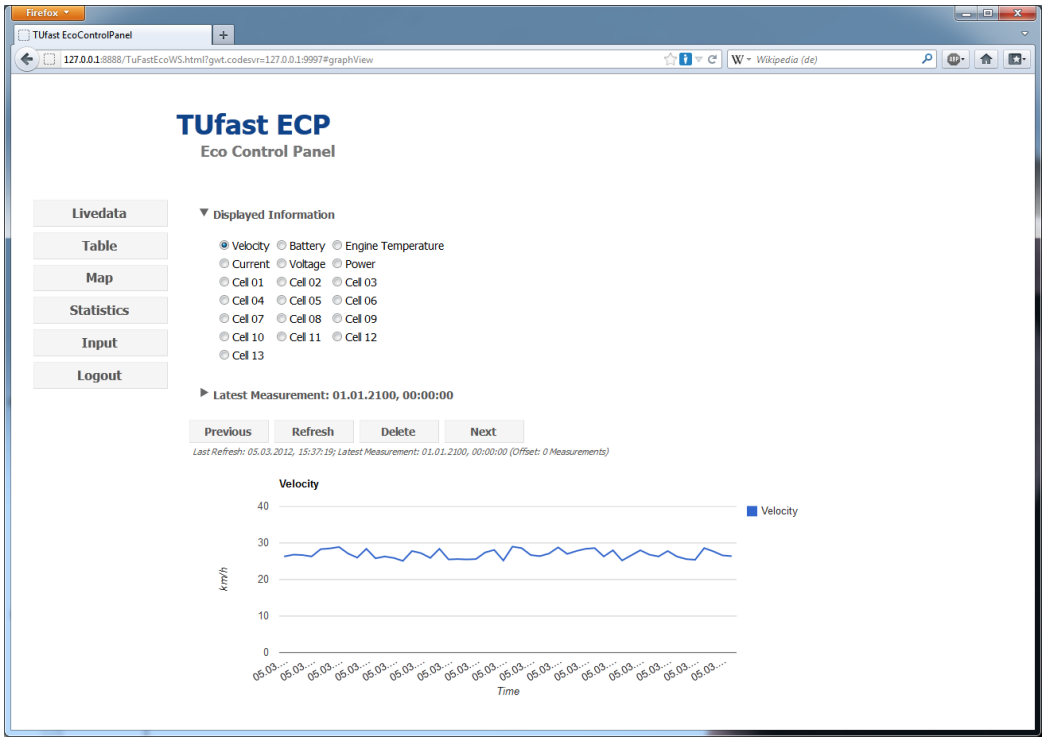


Abbildung 10.4.: Fahrzeugdaten als Graph

Graph

Eine weitere Analyseoberfläche ist in Abbildung 10.4 dargestellt. Diese Übersicht zeigt die letzten 50 Datenpunkte zu einer ausgewählten Achse, z. B. für die Geschwindigkeit, als Graph an. Auch hier können Daten jüngeren Datums von der Anzeige ausgeschlossen werden.

10.3. Eingabemaske

Mit Hilfe der Eingabemaske auf Abbildung 10.5 können Daten als JSON formatierter String, wie er z. B. im Anhang A.2 zu finden ist, manuell auf den Server geladen werden. Diese Eingabemaske kann über den Button Sample auch Beispieldaten eintragen, die dann vom Nutzer angepasst und über den Button Upload in die Datenbank auf dem Server eingetragen werden können.

Mit Hilfe dieser Oberfläche können Daten zu Testzwecken generiert werden und somit die Funktion des Webservices überprüft werden.

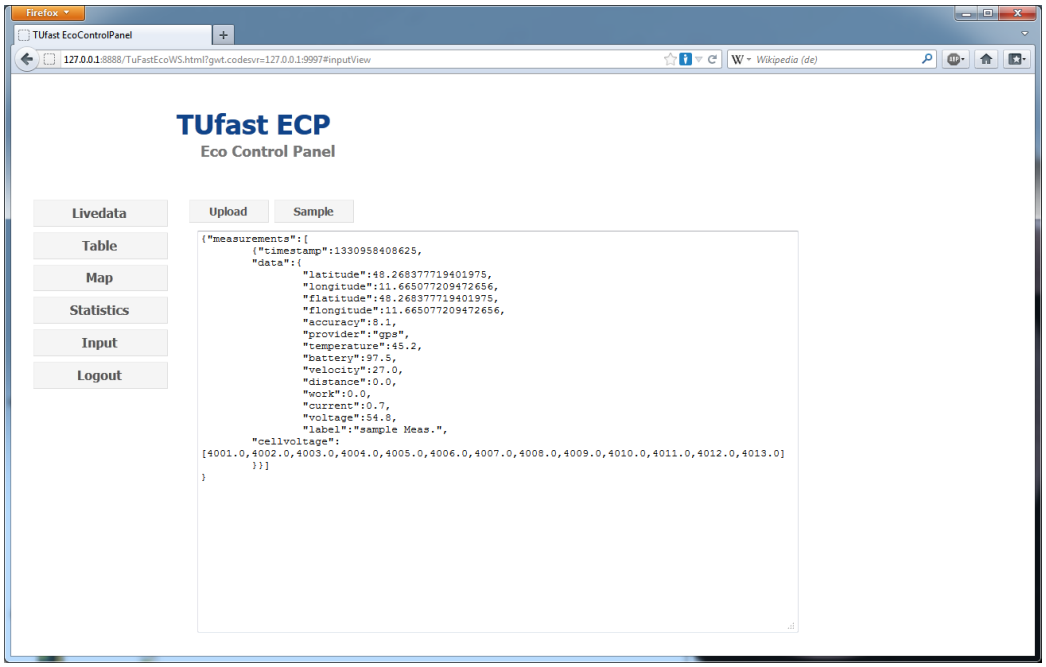


Abbildung 10.5.: Eingabemaske für Fahrzeugdaten als JSON

10.4. Sponsorwebseite

Die Webseite, wie sie in Abbildung 10.6 dargestellt ist, ist im Gegensatz zu den bisher vorgestellten Seiten frei zugänglich und im Design an die Webseite des Vereins angelehnt. Diese Seite soll beim Rennen dazu dienen, Sponsoren und interessierten Besuchern die Möglichkeit zu geben, am aktuellen Renngeschehen teilzunehmen.

Dazu werden links die wichtigsten Fahrdaten angezeigt, daneben findet sich eine Karte in deren Zentrum sich das Fahrzeug aktuell aufhält. Des Weiteren enthält diese Seite eine Übersicht über die aktuellen Twitter Meldungen des TUfast Eco-Teams.

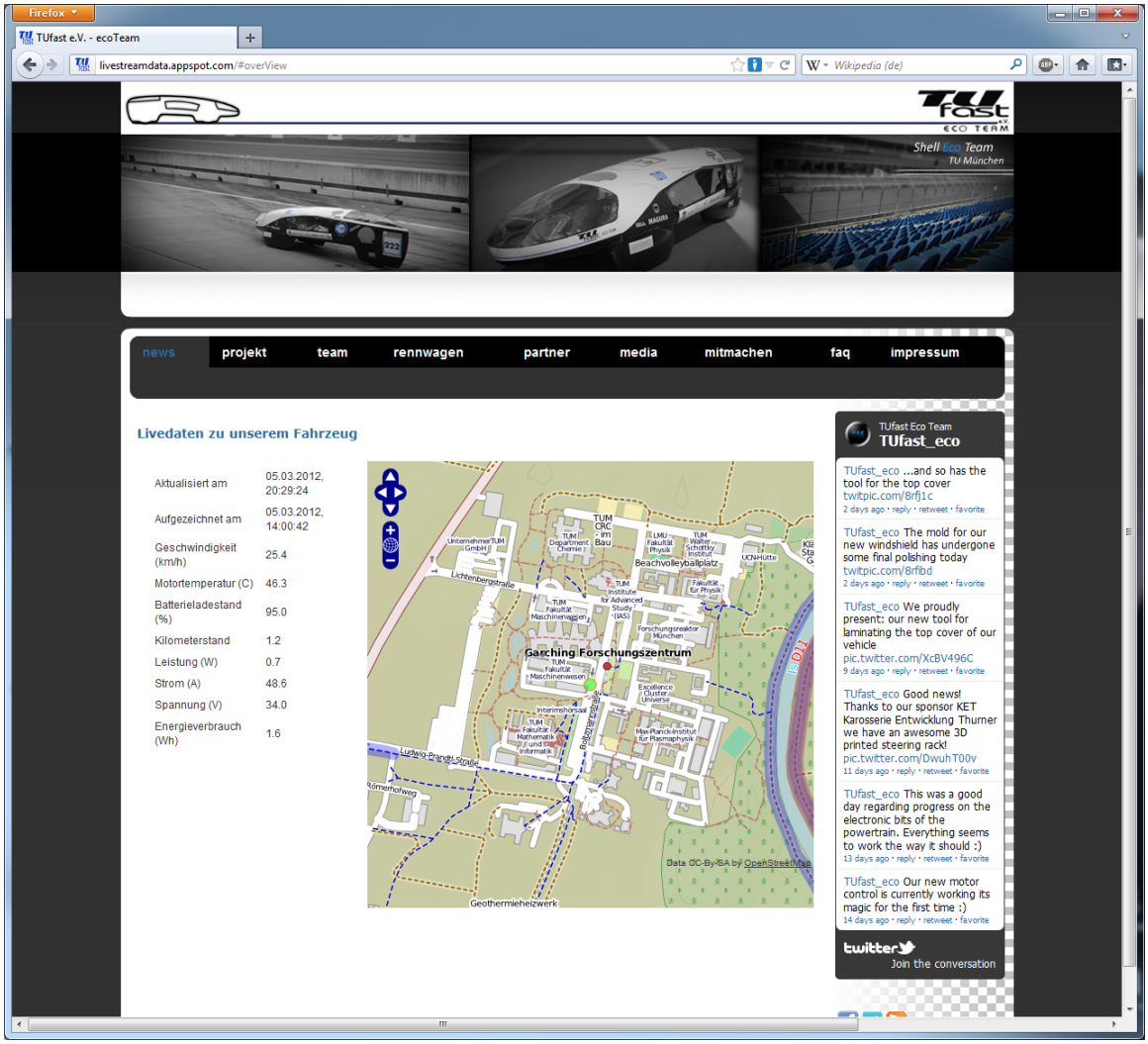


Abbildung 10.6.: Webseite für Sponsoren und Publikum

KAPITEL

ELF

SOFTWARE

11.1. Entwicklungsumgebung

Für die Implementierung des Webservices wurde wie auch bei den vorhergehenden Systemen Eclipse (Version 3.7) eingesetzt. Für die im Webservice verwendeten Frameworks, Google Web Toolkit und Google App Engine existieren Eclipse Plugins, die bei der Entwicklung verwendet wurden [11, 18, 22]. Die Realisierung des Webservices erfolgt in der Programmiersprache Java.

11.2. verwendete Frameworks & Bibliotheken

In der folgenden Aufzählung werden die für diese Anwendung verwendeten Frameworks und Bibliotheken aufgelistet, zusammen mit einer kurzen Begründung für ihren Einsatz.

- **Google Web Toolkit (GWT)** ist ein Framework mit dem sich dynamische Webseiten gestalten lassen, die Nutzer wie normale Anwendungen bedienen können. Die Wahl fiel auf dieses Framework, da es die Realisierung der kompletten Website mit Client- und Serveranteil in Java ermöglicht, wobei der Client nach JavaScript kompiliert wird. Vorteil dieser Kompilierung nach JavaScript ist, dass sehr effizienter und kompakter Code erzeugt wird. Daneben muss der Entwickler nicht auf die Eigenheiten einzelner Browser Rücksicht nehmen, da die Anpassungen ebenfalls vom Framework vorgenommen werden. Auch existiert für das GWT eine große Anzahl an Erweiterungen, z. B. für die Einbindung von Kartenmaterial [21].
- Wie bereits erwähnt ist die **Google App Engine (GAE)** ein Service von Google, bei dem sich Webservices betreiben lassen, wobei je nach Bedarf der Anwendung Rechen- und Speicherkapazität gemietet werden kann. Die GAE wurde ausgewählt, weil sie bis zu einem gewissen Umfang kostenlos ist und kein Administrationsaufwand entsteht [24, 27].

- **Gson 2.1** ist ebenso wie Jackson ein JSON Parser. Für die Server Anwendung wurde allerdings Gson verwendet, da die Konvertierung von JSON zu Java Objekten zwar nicht so performant, aber dafür einfacher ist. Außerdem ist die geringere Performance des Konvertierungsvorgangs auf Serverseite nicht so entscheidend wie auf dem Smartphone, da auf Serverseite eine höhere Rechenkapazität zur Verfügung steht [19].
- **GWT-OpenLayers 0.7** ist eine Portierung der OpenLayers Bibliothek für GWT. Die Aufgabe dieser Bibliothek ist die dynamische Anzeige von Kartenmaterial im Browser. Die Wahl fiel auf diese Bibliothek, da sie im Gegensatz zum Google Kartenmaterial kostenlos ist [41, 47].
- **GWT-Visualization 1.1.2** ist eine Erweiterung für GWT, die es ermöglicht Graphen und Diagramme zu zeichnen. Diese wurde ausgewählt, da sich damit die Sensordaten gut als Graph repräsentieren lassen [17].
- Bei **Objectify 3.1** handelt es sich um eine Bibliothek, die den Zugriff auf die Datenbank der GAE ermöglicht. Die Wahl fiel auf diese Bibliothek, da sich die Datenbank mit dieser Bibliothek einfacher einbinden lässt, als mit den Standardschnittstellen der GAE [46].

11.3. Architektur

Im Folgenden wird die Architektur der Anwendung vorgestellt. Dabei wurde die sogenannte Model View Presenter Architektur verwendet, wie sie auch in der Dokumentation zu GWT vorgeschlagen wird [33].

Client - View & Presenter

Abbildung 11.1 zeigt die Architektur auf Clientseite, wobei auf Clientseite nur View und Presenter vorhanden sind.

Im Paket View finden sich Klassen, die die graphischen Oberflächen repräsentieren.

Dazu gehören die Klasse MenuViewImpl, die sich um die Darstellung der Navigationsleiste kümmert. Daneben die InputDataViewImpl Klasse, die eine Textbox besitzt, mit deren Hilfe Sensordaten in die Datenbank eingetragen werden können. Im Unterpaket measurement finden sich schließlich die vier Klassen, die die Sensordaten als Zusammenfassung, Tabelle, Graph oder auf Karte anzeigen.

Presenter sind Komponenten, die jeweils eine View und deren Logik besitzen. So übernehmen sie die Kommunikation mit anderen Komponenten über den HandlerManager oder die Kommunikation mit dem Server mit Hilfe der Klassen aus dem rpc Paket.

Die Anwendung besitzt nur drei Presenter, da Views, die Sensordaten darstellen, die gleichen Daten verarbeiten, sodass für die Subklassen von AbstractMeasurementsView nur ein gemeinsamer MeasurementPresenter benötigt wird. Neben den Presentern findet sich auch noch eine DataAccessObject Klasse in diesem Paket. Diese ist für die Zwischen-

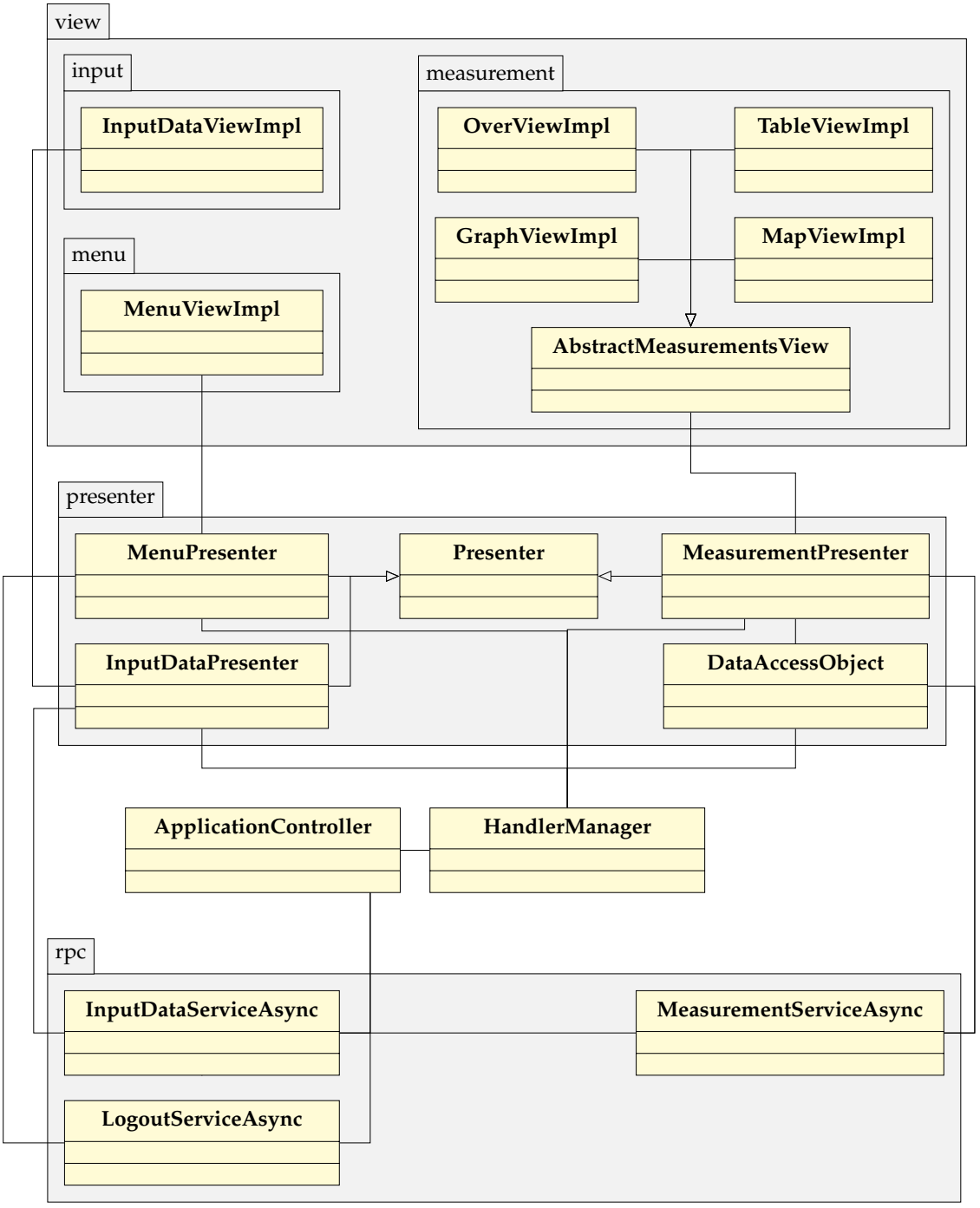


Abbildung 11.1.: Klassendiagramm - Architektur des Webservices auf Clientseite

speicherung von Sensordaten bestimmt, um so bei einem Wechsel der View, nicht jedes mal neue Daten vom Server anfordern zu müssen.

Eine weitere wichtige Komponente ist der AppController. Dieser kümmert sich um die Verwaltung der Browserhistory, d. h. er sorgt für das korrekte Funktionieren der Vor- und Zurücktasten am Browser. Daneben ist der AppController für das Umschalten zwischen einzelnen Seiten zuständig, d. h. er lädt die gewünschten Presenter mit zugehörigen Views und verknüpft die Presenter mit der jeweiligen Service Klasse aus dem rpc Paket. Auf der Abbildung 11.1 wurden Verbindungen zum ApplicationController nicht eingezeichnet (wie z. B. zwischen den Views und dem ApplicationController), um die Übersichtlichkeit zu erhalten.

Die Kommunikation zwischen den Presentern, ApplicationController und DataAccessObject wird über die HandlerManager Klasse abgewickelt. Dabei handelt es sich um einen Event Bus, d. h. eine Klasse, die an diesem Bus angeschlossen ist, kann auf Events eines bestimmten Typs warten und im Falle dieses Events eine Aktion ausführen. So kann zum Beispiel ein MeasurementPresenter auf ein NewMeasurementsEvent warten, das ausgelöst wird, sobald neue Sensordaten am Client eintreffen. Sollten neue Sensordaten eintreffen kann der MeasurementPresenter seine View aktualisieren.

Die Klassen im rpc Paket sind für die Kommunikation zwischen Client und Server zuständig. Jede der dargestellten Klassen besitzt eine entsprechende Klasse auf dem Server, die dann die Verbindung zum Model herstellen.

Server - Model

Der Aufbau des Servers wird im Klassendiagramm in Abbildung 11.2 dargestellt. Dabei ist auch das rpc Paket des Clients dargestellt, das die Kommunikation mit dem Server übernimmt. Dazu besitzt der Client die drei Interfaces InputDataService, LogoutService und MeasurementService, die von den analogen Klassen auf dem Server implementiert werden. Auf Clientseite erfolgt der Zugriff auf diese mit Hilfe der bereits vorgestellten Klassen, die auf -Async enden, wobei die Verknüpfung zwischen den Serviceklassen auf Server und Client vom Framework übernommen wird.

Die einzelnen Services erfüllen jeweils unterschiedliche Zwecke, so können mit Hilfe InputDataServices manuelle Eintragungen in die Datenbank vorgenommen werden, der LogoutService generiert einen Link für die Abmeldung von der Webseite und der MeasurementService ermöglicht die Abfrage von Sensordaten aus der Datenbank.

Die Klasse AcceptData dient zum Empfang der Daten vom Mobiltelefon. Im Gegenzug dient die ProvideData Klasse dazu, Daten für die Sponsorenwebseite zur Verfügung zu stellen.

Die Klasse ObjectifyServiceRegister stellt den Zugang zur Datenbank mittels der bereits vorgestellten Objectify Bibliothek her. Das Datenbankschema selbst ist im Anhang B.2 dargestellt. Bei diesem Datenbankschema wird darauf verzichtet, die einzelnen Werte in separate Felder der Datenbank zu schreiben, sondern es wird lediglich der Aufnahmezeitpunkt (timestamp) separat gespeichert, die restlichen Sensordaten werden als serialisiertes Javaobjekt in die Datenbank geschrieben. Dieses Vorgehen hat den Nachteil, dass

nun Abfragen an die Datenbank nur noch über den timestamp erfolgen können, jedoch nicht mehr über die Einzelwerte der restlichen Sensordaten. Der Vorteil dieses Schemas ist jedoch, dass die Datenbank nur dreispaltig ist, d. h. dieses Struktur hat Vorteile bei der Berechnung der Datenbankkosten der Google App Engine. Dieses Kostenmodell berechnet pro Feld einen Lesezugriff auf die Datenbank, sodass bei der Speicherung der Sensordaten als ein serialisiertes Java Objekt nur ein Lesezugriff berechnet wird. Da eine Abfrage über den timestamp für die Realisierung des Webservices ausreicht, überwiegt der Kostenvorteil, sodass diese Lösung trotz der Reduzierung der Abfragemöglichkeiten in Datenbank realisiert wurde.

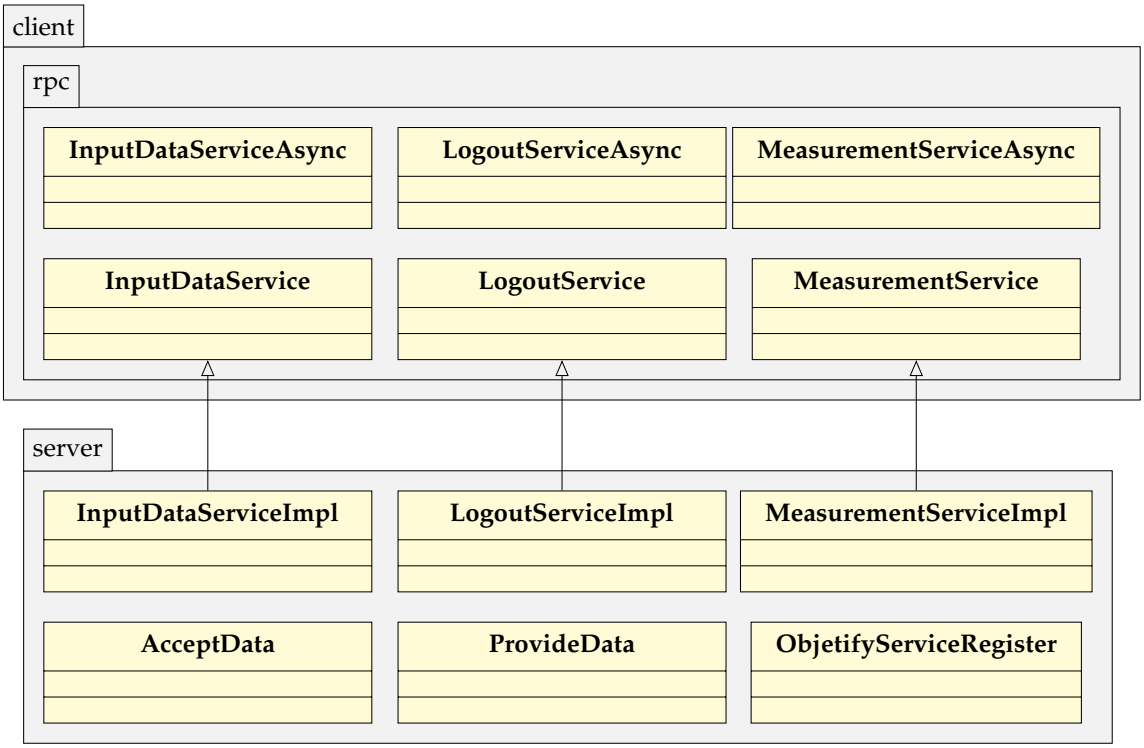


Abbildung 11.2.: Klassendiagramm - Architektur des Webservices auf Serverseite

11.4. Datenfluss

Abbildung 11.3 stellt ein vereinfachtes Schema für die Datenabfrage vom Client an den Server dar. Es ist ausschließlich die Abfrage von Sensordaten am Server abgebildet, jedoch funktioniert die Abfrage anderer Serverdienste analog.

Ein Presenter (auf dem Diagramm das DataAccessObject) stellt eine Anfrage an den einen Service (MeasurementServiceAsync), die Anfrage wird automatisch vom Framework an den entsprechenden Service des Servers weitergeleitet (MeasurementServiceImpl). Hierbei kümmert sich das Framework ohne weiteres Zutun des Programmierers um die Serialiserung von Aufruf- und Rückgabeparametern. Der dargestellte getMeasure-

ment() Aufruf blockiert nicht, d. h. die Anwendung bleibt auf Clientseite auch bedienbar, wenn auf eine Antwort vom Server gewartet wird.

Bei erfolgreicher Rückmeldung der angefragten Daten vom Server löst der Presenter ein entsprechendes Event auf dem HandlerObject aus. Bestimmte Presenter warten nun auf ein Element genau dieses Typs und werden aktiv (in diesem Fall der Measurements-Presenter). Daraufhin wird vom Presenter die zugehörige View, im konkreten Beispiel die TableView, aktualisiert.

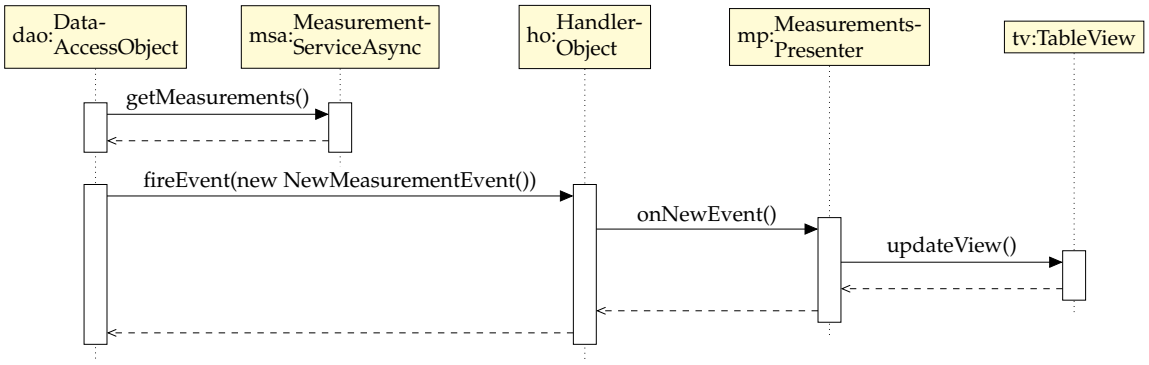


Abbildung 11.3.: Sequenzdiagramm - Aktualisierung von Sensordaten (vereinfacht)

11.5. Anmerkungen zu Sponsorenwebseite

Die Sponsorenwebseite ist als eigenes Projekt in GWT und GAE realisiert, das auf einer separaten Serverinstanz läuft. Zum einen erhöht das die Sicherheit, da diese Seite damit keinen direkten Zugang zur Datenbank besitzt und so Angriffe auf die Datenbank ins Leere laufen. Außerdem wird bei einer erhöhten Anzahl Seitenzugriffe nur die Sponsorenseite nicht aber der Hauptwebservice überlastet.

Architektur und Aufbau der Sponsorenwebseite sind in großen Teilen identisch zum bereits vorgestellten Webseite. Wegen der hohen Ähnlichkeit zur bereits vorgestellten Architektur wird auf die genaue Beschreibung des Aufbaus dieses Services verzichtet und nur eine kurze textuelle Beschreibung dieser Seite gegeben.

Im Unterschied zum Webseite für Teammitglieder besitzt die Sponsorenwebseite nur eine View und auch das Model ist stark vereinfacht, da auf dem Server der Sponsorenwebseite immer nur der aktuellste Datensatz zwischengespeichert wird. Die Abfrage von Sensordaten funktioniert wie folgt. Die Sponsorenwebseite ruft Daten über die Provide-Data Schnittstelle vom Hauptwebservice ab. Dabei kommt wie schon bei der Übertragung von Daten vom Smartphone zum Server, REST, unter Einsatz von HTTP (hier ohne Verschlüsselung) und JSON zum Einsatz. Hierbei wird immer nur der aktuellste Datensatz abgefragt, und auch von diesem nur ein Teil der Daten bereitgestellt. Ein Beispiel für eine solche JSON Nachricht ist im Anhang A.3 dargestellt.

Teil VI.

Qualitätsmanagement

KAPITEL

ZWÖLF

SOFTWARETESTS

12.1. Eingesetzte Werkzeuge

- **Cppcheck 1.53** ist ein Tool, das für die statische Analyse von C/C++ eingesetzt werden kann. Eingesetzt wurde dieses Werkzeug, weil es sich als Plugin in Eclipse integrieren lässt und Empfehlungen für Verbesserungen des Codes gibt [39].
- Bei **FindBugs 2.0** handelt es sich ebenso um ein Tool zur statischen Codeanalyse, allerdings analysiert FindBugs Java Code. Auch dieses Werkzeug wurde verwendet, weil es als Plugin in Eclipse integriert werden kann und Hinweise für Korrekturen des Codes gibt [40].
- **JUnit** ist ein Werkzeug zur Durchführung von Tests in Java. Verwendet wurde dieses Tool, weil es Tests automatisiert ausführt, bereits in Eclipse integriert ist und in den Dokumentationen zum Android und GWT Framework vorgeschlagen wird [25, 26, 44].
- Zusätzlich wurde **Mockito 1.9.0** verwendet. Hierbei handelt es sich um ein Mocking Framework für Java, das dazu verwendet wurde, Verhalten von Klassen bei einer Testdurch-führung zu simulieren [45].

12.2. Testen der Teilsysteme

Testen der Controllersoftware

Der Code dieser Anwendung wurde mit Hilfe von Cppcheck untersucht und die Vorschläge in den Programmcode eingearbeitet.

Für die Durchführung von Tests wurde ein Testsystem mit allen Hardwarekomponenten aufgebaut, die später auch im Fahrzeug verwendet werden und in Abbildung 6.1 dargestellt sind. Mit Hilfe dieser Testumgebung wurden Fehler gesucht und behoben, außerdem

war es möglich die Funktionen der Controllersoftware zu überprüfen.

Testen der Smartphoneanwendung

Auch bei der Smartphone Anwendung kam ein Werkzeug für die statische Codeanalyse zum Einsatz. Bei diesem Tool handelt es sich um FindBugs. Die Vorschläge und Hinweise dieses Tools wurden soweit möglich in den Code miteinbezogen.

Als zweites Werkzeug wurde das JUnit Framework verwendet, wie es auch von Google selbst vorgeschlagen wird. Dazu wurde in Eclipse ein Testprojekt für die Android Software angelegt, das mit Hilfe von JUnit einzelne Teile der Software überprüft. [25].

Da eine vollständige Überprüfung der Software nicht möglich ist, wird im Testprojekt vor allem die korrekte Verarbeitung von Sensordaten überprüft.

Neben den automatisierten Tests wurde die Software auch manuell getestet. Eine Testperson wurde gebeten die App auf seinem Smartphone für mehrere Stunden auszuführen, um das Sammeln von GPS Daten und die Übertragung zum Webservice zu testen. Dabei war der Controller allerdings nicht am Smartphone angeschlossen, da das vollständige Testsystem nicht ohne weiteres im Betrieb transportiert werden kann. In manuellen Versuchen in Verbindung mit dem Testsystem wurde auch die korrekte Anzeige der Sensordaten oder auch das Funktionieren der Telefonfunktion überprüft.

Testen des Webservices

Wie schon bei der Smartphone Anwendung, wurde auch der Code des Webservices mit Hilfe von FindBugs untersucht und verbessert.

Automatisierte Tests mit dem GWT Framework finden nicht in einem eigenen Projekt, sondern in einem Unterordner des Hauptprojekts statt [26].

In diesem Paket finden sich Tests, die mit Hilfe von Mockito und JUnit realisiert wurden. Mit Hilfe dieser automatisierten Tests wird vor allem die korrekte Verarbeitung der Daten auf dem Server überprüft, vom Empfang der JSON formatierten Daten, über die Speicherung in der Datenbank, bis hin zum Auslesen durch den Client.

Ähnlich wie bei der Android App wurde die graphische Anzeige durch manuelle Tests überprüft.

Teil VII.

Zusammenfassung

KAPITEL

DREIZEHN

ZUSAMMENFASSUNG

Das Ziel dieser Bachelorarbeit war der Entwurf und die Realisierung eines Systems, das Daten der Sensorik eines Fahrzeugprototyps, der von Studenten der TU München entwickelt wird, aufzeichnet. Die Daten der Sensorik werden als Instrumententafel für den Fahrer des Prototypen angezeigt und zusätzlich während der Fahrt an einen Webservice übertragen, sodass eine Analyse der Daten noch während der Fahrt möglich ist.

Zur Realisierung dieser Funktionen wurde ein mobiles verteiltes System entworfen und implementiert, das sich aus folgenden Teilsystemen zusammensetzt:

- Das erste dieser drei Systeme ist ein Controllerboard, an das die Steuerung, Sensorik, Motor und Batteriesysteme des Fahrzeugs angeschlossen sind. Mit Hilfe dieses Controllers werden die Sensordaten gesammelt und das Fahrzeug gesteuert.
- Das zweite Teilsystem ist ein Android Mobiltelefon, das per USB mit dem Controller verbunden wird. Dabei erfüllt das Smartphone mehrere Aufgaben zugleich. Es dient als gewöhnliches Telefon zur Sprachkommunikation des Fahrers mit einem Mechaniker. Der Bildschirm des Mobiltelefons dient dem Fahrer als Instrumententafel. Zusätzlich werden die angezeigten Daten um GPS Positionen erweitert und dann zu einem Webservice geschickt.
- Teilsystem Nummer drei ist schließlich der Webservice, der die empfangenen Daten archiviert und darstellen kann.

Das komplette System ist zum aktuellen Zeitpunkt als funktionierender Prototyp aufgebaut. Dieser Prototyp besteht aus einem Nachbau des Antriebsstrangs des Fahrzeugs, also dem Elektromotor, der Motorsteuerung und den Akkus zusammen mit dem Batteriemanagementsystem. An diesen Antriebsstrang wurden die drei oben beschriebenen Teilsysteme angeschlossen und erfolgreich getestet.

KAPITEL

VIERZEHN

AUSBLICK & BEWERTUNG

Die Entwicklung des Systems ist mit dem Aufbau des funktionierenden Antriebsstrangs abgeschlossen. Der nächste Arbeitsschritt wird die Integration des System in das Fahrzeug sein. Diese wird voraussichtlich im kommenden Monat erfolgen. Der aktuelle Zeitplan des TUfast Eco-Teams sieht die Fertigstellung des Fahrzeugs gegen Ende April vor, somit bleibt noch circa ein Monat vor dem Shell Eco-marathon, der vom 17. - 19. Mai in Rotterdam stattfindet.

Dieser verbleibende Monat wird dazu genutzt werden Testfahrten durchzuführen, um noch letzte Fehler im System zu finden. Außerdem besteht vermutlich noch Optimierungspotential bei der Ansteuerung des Motors, die bis jetzt nur sehr rudimentär implementiert wurde. Diese Anpassungen können jedoch erst vorgenommen werden, wenn das Fahrzeug fahrbereit ist.

Anhang

ANHANG

A

DATENFORMATE

A.1. Datenformat für Austausch zwischen Controller & Smartphone

In den folgenden Tabellen wird der Aufbau der Nachrichten beschrieben, die vom Controller an das Smartphone geschickt werden. Dazu werden die Daten in ein Array geschrieben, das aus 46 genau ein Byte großen Zellen aufgebaut ist.

Sensordatenformat

Byte	Funktion
0	Nachrichtentyp (als Wert 1 möglich)
1	Zähler für Drehencoder (Werte 0 - 256)
2	Schalterzustände für Start, Drehencoder und Sperrzustand (je ein Bit)
3	Eingestellter Strom (Werte von 0 - 99)
4 - 5	Gemessene Temperatur, höherwertiges Byte in 4 (Werte von 0 - 1024)
6 - 7	Gemessener Strom, höherwertiges Byte in 6 (Werte von 0 -1024)
8 - 9	Gemessener Batteriestand, höherwertiges Byte in 8, *nicht verwendet, da noch fehleranfällig, eventuell künftig Kalibrierung möglich*
10 - 11	Gemessene Motordrehzahl, höherwertiges Byte in 12
12 - 37	Zellspannungen Zelle 1 - 13, höherwertige Byte jeweils voranstehend
38 - 39	Gemessene Temperatur am BMS, höherwertiges Byte in 40, *nicht verwendet, da ungenauer als Temperatursensor*
40 - 41	Gemessene Stromstärke am BMS, höherwertiges Byte in 42, *nicht verwendet, da ungenauer als Stromsensor*
42 - 43	*Reserviert für Daten der Motorsteuerung für künftige Optimierungen*
44 - 45	*Reserviert für Daten der Motorsteuerung für künftige Optimierungen*

Debugdatenformat

Byte	Funktion
0	Nachrichtentyp (Wert 2 für Daten von Motorsteuerung zu Controller, Wert 3 für umgekehrte Kommunikationsrichtung)
1 - 45	Nachricht encodiert als String (ASCII, 0 terminiert)

A.2. Datenformat für Austausch zwischen Smartphone & Webservice

Dieses Beispiel zeigt das Format der Sensordaten, die als Datenteil eines HTTP Post Pakets von der Smartphone App an die Unteradresse `/rest/accept` des Webservices geliefert wird.

```
{
  "measurements":[{
    "timestamp":1330958408625,        // Zeitpunkt der Aufnahme
    "data":{
      "latitude":48.268,              // Breitengrad ungefiltert
      "longitude":11.665,             // Laengengrad ungefiltert
      "flatitude":48.260,             // Breitengrad gefiltert
      "flongitude":11.660,            // Laengengrad gefiltert
      "accuracy":8.1,                 // Genauigkeit der Positionsdaten
      "provider":"gps",               // Quelle der Positionsdaten
      "temperature":45.2,             // Motortemperatur
      "battery":97.5,                 // Batterieladestand
      "velocity":27.0,                // Geschwindigkeit
      "distance":0.0,                 // Gefahrene Strecke
      "work":0.0,                     // Verrichtete Arbeit
      "current":0.7,                  // Strom
      "label":"sample Measurement",   // Quelle der Aufnahme
      "cellvoltage":[                 // Array mit Zellspannungen
        4001.0,
        4002.0,
        4003.0,
        4004.0,
        4005.0,
        4006.0,
        4007.0,
        4008.0,
        4009.0,
        4010.0,
        4011.0,
        4012.0,
        4013.0
      ]
    }
  }]
}
```

A.3. Datenformat für Austausch zwischen Webservice & Sponsorenwebseite

Dieses Beispiel zeigt das Format der Sensordaten, die der Webservice unter der Adresse `/unrestricted/providedata` für die Sponsorwebseite zur Verfügung stellt.

```
{
  "measurements":[{
    "timestamp":1330958408625,        // Zeitpunkt der Aufnahme
    "data":{
      "flatitude":48.260,             // Breitengrad gefiltert
      "flongitude":11.660,            // Laengengrad gefiltert
      "temperature":45.2,             // Motortemperatur
      "battery":97.5,                 // Batterieladestand
      "velocity":27.0,                // Geschwindigkeit
      "distance":0.0,                 // Gefahrene Strecke
      "work":0.0,                     // Verrichtete Arbeit
      "current":0.7,                  // Strom
      "voltage":52.2,                 // Gesamtspannung
    }
  }]
}
```

ANHANG

B

DATENBANKSCHEMATA

B.1. Datenbankschemata Android

Dieser Abschnitt zeigt die Schemata der beiden Datenbanken, die in der Android App verwendet werden.

Datenbankschema für Sensordaten

Name	Datentyp	Beschreibung
id	INTEGER	Primärschlüssel (autoincrement)
timestamp	LONG	Zeitpunkt der Datenaufzeichnung
latitude	DOUBLE	Breitengrad (ungefiltert)
longitude	DOUBLE	Längengrad (ungefiltert)
fil_latitude	DOUBLE	Breitengrad (gefiltert)
fil_longitude	DOUBLE	Längengrad (gefiltert)
accuracy	DOUBLE	Abweichung der Positionsdaten in m
provider	STRING	Ursprung der Positionsdaten z. B. GPS
temperature	DOUBLE	Temperatur am Temperatursensor des Controllers in ° C
battery	DOUBLE	Ladestand der Batterie in Prozent
velocity	DOUBLE	Aktuelle Geschwindigkeit in km/h
distance	DOUBLE	Zurückgelegte Strecke in km
work	DOUBLE	Verrichtete Arbeit in Wh
current	DOUBLE	Stromstärke am Stromsensor des Controllers in A
label	STRING	Quelle der Aufnahme z. B. „Smartphone“
cell0 - cell12	DOUBLE	Spannung der 13 Akkuzellen in mV

Datenbankschema für Debugdaten

Name	Datentyp	Beschreibung
id	INTEGER	Primärschlüssel (autoincrement)
timestamp	LONG	Zeitpunkt der Datenaufzeichnung
msgtype	STRING	Ursprung der Nachricht (Controller oder Motorsteuerung)
text	STRING	Nachricht

B.2. Datenbankschema Webservice

Dieser Abschnitt zeigt das Datenbankschema des Webservices. Um Anfragen zu beschleunigen besitzt das Feld timestamp einen Index.

Name	Datentyp	Beschreibung
id	LONG	Primärschlüssel
timestamp	LONG	Zeitpunkt der Datenaufzeichnung
data	BINARY	Sensordaten als serialisiertes Java Objekt

ANHANG

C

DATENTRÄGER

Auf diesem Datenträger befindet sich der Quellcode aller beschriebenen Anwendungen.

ABBILDUNGSVERZEICHNIS

LITERATURVERZEICHNIS

[1] Arduino. *Arduino ADK*. http://arduino.cc/en/Main/ArduinoBoardADK. zuletzt abgerufen am 13.03.2012.

[2] Arduino. *Bounce Library*. http://www.arduino.cc/playground/Code/Bounce. zuletzt abgerufen am 13.03.2012.

[3] Arduino. *Eclipse Integration*. http://arduino.cc/playground/Code/Eclipse. zuletzt abgerufen am 13.03.2012.

[4] Arduino. *Frequently Asked Questions*. http://arduino.cc/en/Main/FAQ. zuletzt abgerufen am 13.03.2012.

[5] Arduino. *Libraries for Arduino*. http://arduino.cc/playground/Main/LibraryList. zuletzt abgerufen am 13.03.2012.

[6] Bourns. *PEC12 - 12 mm Incremental Encoder*. http://www.bourns.com/pdfs/pec12.pdf. zuletzt abgerufen am 13.03.2012.

[7] Brügge, B. and Dutoit, A. H. *Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java*. Pearson Education, München, 2004.

[8] Crockford, D. *The application/json Media Type for JavaScript Object Notation (JSON)*. http://www.ietf.org/rfc/rfc4627.txt?number=4627. zuletzt abgerufen am 13.03.2012.

[9] CyanogenMod. *Building Kernel from source*. http://wiki.cyanogenmod.com/wiki/Building_Kernel_from_source#Setup_Repo. zuletzt abgerufen am 13.03.2012.

[10] CyanogenMod. *Nexus S: Full Update Guide*. http://wiki.cyanogenmod.com/wiki/Nexus_S:_Full_Update_Guide. zuletzt abgerufen am 13.03.2012.

[11] Eclipse. *Eclipse Download*. http://www.eclipse.org/downloads/. zuletzt abgerufen am 13.03.2012.

[12] Elmo Motion Control. *Solo Whistle Digital Servo Drive Installation Guide*. http://www.elmomc.com/support/manuals/MAN-SOLWHIIG.pdf. zuletzt abgerufen am 13.03.2012.

[13] Fielding, R. *Architectural Styles and the Design of Network-based Software Architectures*.

http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. zuletzt abgerufen am 13.03.2012.

[14] Google. *ADT Plugin for Eclipse*. http://developer.android.com/sdk/eclipse-adt.html. zuletzt abgerufen am 13.03.2012.

[15] Google. *Android 2.3 Platform*. http://developer.android.com/sdk/android-2.3.html. zuletzt abgerufen am 13.03.2012.

[16] Google. *Android Open Accessory Development Kit*. http://developer.android.com/guide/topics/usb/adk.html. zuletzt abgerufen am 13.03.2012.

[17] Google. *Google API Libraries for Google Web Toolkit*. http://code.google.com/p/gwt-google-apis/. zuletzt abgerufen am 13.03.2012.

[18] Google. *Google App Engine Downloads*. http://code.google.com/appengine/downloads.html. zuletzt abgerufen am 13.03.2012.

[19] Google. *google-gson*. http://code.google.com/p/google-gson/. zuletzt abgerufen am 13.03.2012.

[20] Google. *google-guice*. http://code.google.com/p/google-guice/. zuletzt abgerufen am 13.03.2012.

[21] Google. *Google Web Toolkit*. http://code.google.com/webtoolkit/. zuletzt abgerufen am 13.03.2012.

[22] Google. *Google Web Toolkit Downloads* . http://code.google.com/webtoolkit/download.html. zuletzt abgerufen am 13.03.2012.

[23] Google. *Intent*. http://developer.android.com/reference/android/content/Intent.html. zuletzt abgerufen am 13.03.2012.

[24] Google. *Safety Quotas and Billable Quotas*. http://code.google.com/appengine/docs/quotas.html#Safety_Quotas_and_Billable_Quotas. zuletzt abgerufen am 13.03.2012.

[25] Google. *Testing Fundamentals*. http://developer.android.com/guide/topics/testing/testing_android.html. zuletzt abgerufen am 13.03.2012.

[26] Google. *Unit Testing GWT Applications with JUnit*. http://code.google.com/webtoolkit/doc/latest/tutorial/JUnit.html. zuletzt abgerufen am 13.03.2012.

[27] Google. *What is Google App Engine?* http://code.google.com/appengine/docs/whatisgoogleappengine.html. zuletzt abgerufen am 13.03.2012.

[28] U. Hammerschall. *Verteilte Systeme und Anwendungen*. Pearson Education, München, 2005.

[29] MAXIM. *+5V-Powered, Multichannel RS-232 Drivers/Receivers*. http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf. zuletzt abgerufen am 13.03.2012.

[30] Nazaruk, I. *Using internal (com.android.internal) and hidden (@hide) APIs [Part 1, Introduction]*. https://devmaze.wordpress.com/2011/01/18/

using-com-android-internal-part-1-introduction/. zuletzt abgerufen am 13.03.2012.

[31] O2Micro. *OZ890 Datasheet*. 2009.

[32] Pololu. *ACS715 Current Sensor Carrier 0 to 30A*. http://www.pololu.com/catalog/product/1186. zuletzt abgerufen am 13.03.2012.

[33] Ramsdale, C. *Google Web Toolkit Large scale application development and MVP*. http://code.google.com/webtoolkit/articles/mvp-architecture.html. zuletzt abgerufen am 13.03.2012.

[34] Reenskaug, T. *Models- Views- Controllers*. http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf. zuletzt abgerufen am 13.03.2012.

[35] Schmidt, M. *Arduino Ein schneller Einstieg in die Microcontroller-Entwicklung*. dpunkt.verlag, Heidelberg, 2012.

[36] Shell. *History of Shell Eco-marathon*. http://www.shell.com/home/content/ecomarathon/about/history/. zuletzt abgerufen am 13.03.2012.

[37] Shell. *Shell Eco-marathon Official Rules 2012 Chapter I*. http://www-static.shell.com/static/ecomarathon/downloads/2012/SEM_global_Rules_2012_chapter_I.pdf. zuletzt abgerufen am 13.03.2012.

[38] Texas Instruments. *LM35 Precision Centigrade Temperature Sensors*. http://www.ti.com/lit/ds/symlink/lm35.pdf. zuletzt abgerufen am 13.03.2012.

[39] *Cppcheck A tool for static C/C++ code analysis*. http://cppcheck.sourceforge.net/. zuletzt abgerufen am 13.03.2012.

[40] *FindBugs - Find Bugs in Java Programs*. http://findbugs.sourceforge.net/. zuletzt abgerufen am 13.03.2012.

[41] *GWT-OpenLayers*. https://bitbucket.org/gwtopenlayers/gwt-openlayers. zuletzt abgerufen am 13.03.2012.

[42] *Jackson High-performance JSON processor.* http://jackson.codehaus.org/. zuletzt abgerufen am 13.03.2012.

[43] *JKalman*. http://sourceforge.net/projects/jkalman/. zuletzt abgerufen am 13.03.2012.

[44] *JUnit.org Resources for Test Driven Development*. http://www.junit.org. zuletzt abgerufen am 13.03.2012.

[45] *mockito*. http://code.google.com/p/mockito/. zuletzt abgerufen am 13.03.2012.

[46] *objectify-appengine*. http://code.google.com/p/objectify-appengine/. zuletzt abgerufen am 13.03.2012.

[47] *OpenLayers: Free Maps for the Web* . http://openlayers.org/. zuletzt abgerufen am 13.03.2012.

[48] *roboguice*. http://code.google.com/p/roboguice/. zuletzt abgerufen am 13.03.2012.

[49] Xilinx. *Rotary Encoder Interface for Spartan-3E Starter Kit*. `http://www.xilinx.com/products/boards/s3estarter/files/s3esk_rotary_encoder_interface.pdf`. zuletzt abgerufen am 13.03.2012.