TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik

CSR-12-03

Generische Datenerfassung und Aufbereitung im Kontext verteilter, heterogener Sensor-Aktor-Systeme

René Bergelt · Matthias Vodel · Wolfram Hardt

August 2012

Chemnitzer Informatik-Berichte

Generische Datenerfassung und Aufbereitung im Kontext verteilter, heterogener Sensor-Aktor-Systeme

René Bergelt, Matthias Vodel, Wolfram Hardt
berre|vodel|hardt@informatik.tu-chemnitz.de

Technische Universität Chemnitz
Fakultät für Informatik, Professur Technische Informatik

Kurzfassung. Die vorgestellte Arbeit präsentiert ein geschlossenes Konzept für die synchronisierte Erfassung, Verarbeitung und Aufbereitung beliebiger Sensor-Informationen. Es ist nun möglich, heterogene Sensornetze sowie dedizierte, autarke Messsysteme zeitlich zu koordinieren und entsprechend in Relation zu setzen. Auf Basis von XML erfolgt die ganzheitliche Beschreibung des Monitoring-Szenarios und die Einordnung der einzelnen Datensätze. Die Informationen können nun in beliebigen Ausgabeformaten anwendungsspezifisch definiert und visualisiert werden. Zusätzlich ermöglichen Mechanismen zur gezielten Messwert-Vorverarbeitung und -Filterung eine Senkung des benötigten Datenvolumens. Die Funktionalität des vorgestellten GREASE-Frameworks[1] wird am Beispiel eines geschlossenen Test-Systems aus dem Automotive-Bereich evaluiert. Dabei wird die vorhandene Sensorik eines Kfz-Bordnetzes mit einem zusätzlichen Netzwerk aus Sensorknoten verfeinert. Die korrelierten Daten werden anschließend für die Visualisierung mittels Google Earth, jBEAM und anderen Lösungen aufbereitet.

1 Motivation

Im Rahmen aktueller Forschungen auf dem Gebiet drahtloser Sensornetzwerke existieren zahlreiche Projekte, welche mit unterschiedlichen, proprietären Hardwareplattformen arbeiten. Das Ziel dieser Tätigkeiten ist hierbei der Aufbau einer umfassenden Datenbasis, die Informationen verschiedener Art versammelt und es ermöglichen soll, Erkenntnisse aus der Gesamtheit der Ergebnisse zu erlangen. Dazu sollen die verschiedenen Daten unabhängig ihrer Herkunft verschmolzen, gemeinsam untersucht und analysiert werden können. Dieses Ziel ist schematisch in Abbildung 1 dargestellt. Derzeit besitzt die Vielzahl der Messszenarien jedoch meist anwendungsspezifische und voneinander unabhängige Verarbeitungsprozesse für die Datenaufnahme, -speicherung und -auswertung. Zwischen den autarken Messsystemen existieren dabei keinerlei Synchronisationsmöglichkeiten. Infolgedessen ist eine zielgerichtete, detaillierte Aufbereitung der Informationen innerhalb einer gemeinsamen Datenbasis nicht ohne Weiteres möglich. Die Datensätze der jeweili-

[1] GREASE ... Generische, rekonfigurierbare Erfassung und Auswertung von Sensordaten

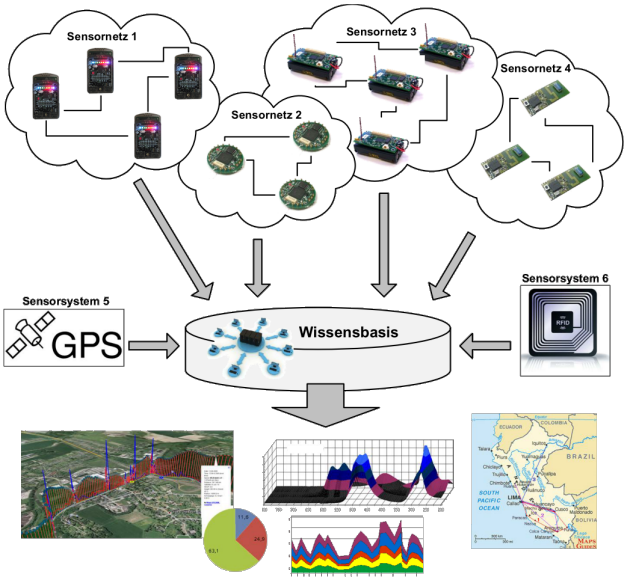


Abbildung 1: Integration unterschiedlicher Sensornetze durch ein synchronisiertes Verarbeitungsframework

gen Sensorplattformen verfügen dazu über keinen einheitlichen Primärschlüssel, wodurch sie nicht eindeutig in Relation zueinander gesetzt werden können.

2 Stand der Technik

Es existieren bereits diverse Systeme zur Messdatenaufnahme und Messdatenauswertung am Markt, die jedoch meist funktionalen oder systembedingten Einschränkungen unterliegen. Viele Anbieter von Sensoren liefern so beispielsweise speziell angepasste Auslesesoftware, die auf Geräte einer bestimmten Produktreihe beschränkt ist. Andere Sensorsysteme überlassen das konkrete Auslesen der Daten ganzheitlich dem Anwender und bieten keinerlei Unterstützung bei der Aufbereitung. Bei produktgebundenen Auswertungstools muss der Hersteller die Messdatenaufnahme systembedingt so allgemein wie möglich gestalten. Für die automatisierte beziehungsweise teilautomatisierte Aufbereitung der Messreihen müssen spezielle Anwendungen hinzugezogen werden, um den Nutzer anwendungsspezifisch zu unterstützen. Bekannte Vertreter dieser Art von Messanalysesoftware sind LabView von National Instruments[2], jBEAM der Firma AMS[3] sowie die Datenanalyse-Software FlexPro[4]. Die Anwendungen ermöglichen sowohl die Auswertung

[2] www.ni.com/labview/d/
[3] www.jbeam.de/unternehmen/produkte/jbeam.html
[4] www.weisang.com/

von importierten Offline-Daten als auch die Live-Analyse von Messdaten, die unmittelbar in das System einfließen. jBEAM und LabView sind hierbei plattformunabhängig einsetzbar und unterstützen eine Vielzahl von Datenformaten und Schnittstellen. Durch den modularen Aufbau und durch die Implementierung des ASAM-Standards[5] erlaubt jBEAM darüber hinaus eine einfache Erweiterung durch eigene Komponenten. FlexPro bietet eine Vielzahl zusätzlicher Module für visuell ansprechende Präsentationen der Messergebnisse.
Der entscheidende Nachteil aller vorgestellten Anwendungen resultiert jedoch aus der Tatsache, dass diese aufgrund der Funktionsvielfalt und den damit einhergehenden Systemvoraussetzungen nicht für die reine Datenaufnahme in ressourcenbeschränkten Hardware-Umgebungen geeignet sind. Sie kommen meist in einer nachfolgenden Instanz zum Einsatz, um die bereits aufgenommen Daten zu verarbeiten. Energieoptimierte und platzsparende Systeme, die nur Daten sammeln, aber nicht auswerten sollen, können diese Anwendungen nicht nutzen. Es fehlt folglich an Hilfsmitteln und Standards, um beliebige Sensorkonfigurationen und -informationen generisch in eine gemeinsame Verarbeitungsinstanz zu leiten, welche die Daten anschließend analysieren und auswerten kann. Dies führt dazu, dass nahezu jedes Sensorprojekt unterschiedliche, meist proprietäre Softwarelösungen nutzt, die zueinander inkompatibel sind. Änderungen in den Konfigurationen des Messszenarios sind infolgedessen nur mit zusätzlichem Zeit- und Arbeitsaufwand realisierbar.

3 Konzept

An dieser Stelle setzt das vorgestellte Konzept an und bietet dem Nutzer nun ein modulares Software-Framework für die synchronisierte Erfassung beliebiger Sensorinformationen. Der Name des vorgestellten Frameworks lautet *GREASE* (Generic Reconfigurable Evaluation and Analysis of Sensor data; Generische, rekonfigurierbare Erfassung und Auswertung von Sensordaten) und verdeutlicht, dass der generische Aspekt und die einfache Rekonfigurierbarkeit eines Messaufbaus im Vordergrund stehen. Um diese Funktionalität zu gewährleisten, müssen zunächst die zu erfüllenden, zentralen Anforderungen definiert werden.

3.1 Anforderungen

Primäres Ziel ist es, eine Verarbeitungsumgebung zu schaffen, die flexibel auf Veränderungen in der Konfiguration und in den Analyseanforderungen der Messumgebung reagieren kann. Darüber hinaus soll die generische Messwertaufnahme und -auswertung ressourcenschonend umgesetzt werden. Der Verarbeitungsprozess muss dabei in zwei räumlich, zeitlich und plattformtechnisch getrennte Phasen unterteilbar sein. Auf der einen Seite arbeiten

[5] ASAM ... Association for Standardisation of Automation and Measuring Systems, www.asam.net

alle Komponenten zur Datenaufnahme und auf der anderen alle für die Datenaufbereitung relevanten Module. Dies steht im Gegensatz zu vorhandenen Lösungen, in denen diese klare Trennung entweder nicht existiert oder verschiedene Konfigurationen gänzlich unterschiedliche Methoden verwenden, die nicht ohne weiteres zusammengeführt werden können. Der Fokus des GREASE-Frameworks liegt somit nicht auf der eigentlichen Datenaufnahme oder –auswertung, wie bei den vorgestellten Applikationen, sondern dem standardisierten Transport und der Synchronisation von beliebigen Messdaten. Dieser Ansatz führt zum Entstehen einer universellen Datenbasis aus Versuchen, da die Datenauswertung bei der Messdatenaufnahme nicht im Vordergrund steht und somit keine Annahmen über den Verwendungszweck der Daten gemacht werden müssen. Der Vorteil ist, dass eine Änderung oder Erweiterung der Auswertung keinerlei Einfluss auf die eigentliche Aufnahme der Daten hat. Dies ist bei schwer zugänglichen oder aufwändig zu konfigurierenden Messständen ein erheblicher Vorteil. Des Weiteren sollen alle Abläufe, auf die der Nutzer über eine graphische Oberfläche Zugriff hat, auch automatisiert bzw. teilautomatisiert gestartet und überwacht werden können. Auch hier hebt sich das entwickelte Framework von den bereits genannten Anwendungen ab, bei denen dies aufgrund des Anwendungsfokus nicht konsequent umgesetzt wurde. Um aber kontinuierlich und unbeaufsichtigt Sensordaten aufzunehmen und auszuwerten zu können, ist diese Funktionalität häufig zwingend erforderlich. Alle zentralen Anforderungen können wie folgt aufgeschlüsselt werden:

- Synchronisation unterschiedlicher, autarker Sensorsysteme
- Modulare Erweiterbarkeit / einfache Modifizierbarkeit
- Nutzung von XML (Extendible Markup Language) als Austauschformat anstatt proprietärer Formate
- Auswertung der Daten aus einer Datenbank, aber auch direkt aus Log-Dateien
- Grafische Konfigurationsoberfläche
- Automatisierte Auswertungsmechanismen

Das Framework soll hierbei als Vermittler zwischen anwendungsspezifischen Komponenten fungieren, jedoch selbst keine anwendungsspezifische Logik besitzen, um universell einsetzbar zu sein.

3.2 Struktur

Die Verarbeitungsfolge von Messdaten innerhalb von Messszenarien ist in Abbildung 2 dargestellt. Die Anforderungen an ein generisches Framework, dass flexibel und anpassungsfähig sein soll, ist nun dergestalt, dass jeder dieser Schritte so dynamisch wie möglich gestaltet werden muss. Dies gelingt durch den vollständig modularen Aufbau

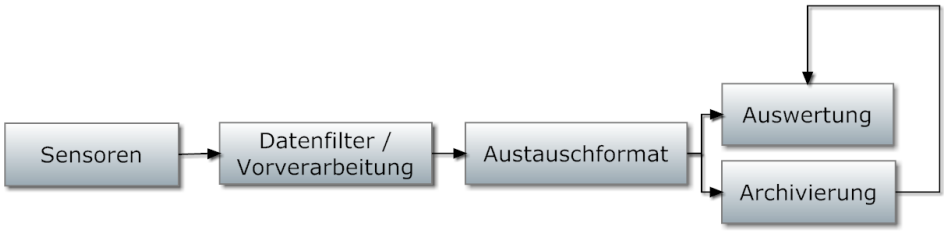


Abbildung 2: Fluss der Messdaten innerhalb des Frameworks

des GREASE-Frameworks. Die Umgebung bedient sich dabei der sogenannten Stundenglasarchitektur (Ende-zu-Ende-Prinzip), die eine maximale Interoperabilität zwischen verschiedenen Komponenten ermöglicht [SRC84]. Dies bedeutet, dass auf der Eingangs- (Datenaufnahme) als auch auf der Ausgangsseite (Datenauswertung) eine große Diversität vorliegt, die durch einen uniformen Mittelteil verbunden wird. Das gesamte Framework besteht dabei im Kern aus der Definition von Objekttypen und Schnittstellen und einer Plugin-Architektur. Dabei existieren fünf Haupttypen von Plugins, drei Typen auf Seiten der Datenaufnahme und zwei Typen auf Seiten der Datenauswertung. Erstgenannte umfassen SensorReader, DataFilter und SensorLogWriter wohingegen letztgenannte aus SensorLogReader und LogReader bestehen. Die Koordinierung erfolgt auf beiden Seiten durch einen sogenannten PluginController. Jeder Plugintyp erfüllt eine spezielle Funktion der in Abbildung 2 dargestellten Schritte. Bei SensorReadern handelt es sich um den speziellsten Plugintyp, da diese die Übersetzung von Messdaten von Sensoren außerhalb des Frameworks in framework-konforme Daten übernehmen. SensorReader können Messdaten in Messdatenpaketen (SensorData-Objekte) zusammenfassen, die aus ein oder mehren Werten, SensorValues[6], bestehen. Diese Messdatenpakete können dann an den PluginController übergeben werden. Sind an diesem DataFilter-Plugins angemeldet, so wird jedes empfangene SensorData-Objekt zuerst an diese übergeben, damit sie die Messdaten filtern, bearbeiten oder aussortieren können. Über DataFilter-Plugins ist somit eine Vorsortierung und Vorverarbeitung möglich. Um die aufgenommenen Messdaten in ein austauschbares Format zu übertragen, muss genau ein SensorLogWriter-Plugin am Controller angemeldet sein, welches die empfangenen, eventuell verarbeiteten Daten speichert. Dazu wird es die framework-bekannten Datenformate in ein plugin-abhängiges Datenformat übertragen und auf einem Speichermedium ablegen (die sogenannte Persistenzebene). Diese gespeicherten Daten können dann vom Messstand abgerufen und an den Teil des Frameworks weitergegeben werden, der für die Datenauswertung zuständig ist. Für eine Rückübertragung der Daten aus dem Datenaustauschformat in framework-Datentypen muss zu jedem SensorLogWriter-Plugin ein entsprechendes Gegenstück in Form eines SensorLogReader-Plugins existieren, welches die zuvor gespeicherten Daten einliest und zurück konvertiert. Die Koordinierung erfolgt dabei wieder über einen PluginController, welcher die Messdaten, die nun wieder in bekannten Datentypen vorliegen, an durch den Nutzer definierte Auswerte-Plugins (LogReader-Plugins) weitergibt. Diese Plugins können die Messdaten dann auswerten, visualisieren oder archivieren. Auch eine Um-

[6]Zur Beschreibung eines Messwerts sind mindestens Bezeichnung, Datentyp sowie Datenwert notwendig

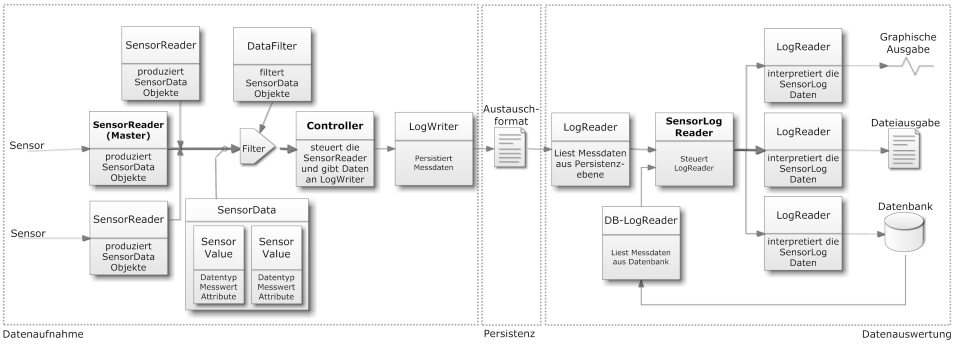


Abbildung 3: Struktur des GREASE-Frameworks
Deutlich zu erkennen ist die Zweiteilung in Datenaufnahme (links) und Datenauswertung (rechts)

wandlung in andere Datenformate zwecks Weiterverarbeitung in Auswertungsanwendungen wie jBEAM ist möglich. Bei der Archivierung in eine Datenbank besteht der Vorteil der modularen Architektur vor allem darin, dass ein entsprechendes SensorLogReader-Plugin die archivierten Daten zu einem späteren Zeitpunkt wieder auslesen und den Auswerte-Plugins zur Verfügung stellen kann. Diese wissen somit nicht, woher die auszuwertenden Daten stammen und dies ist für ihre Funktion auch nicht notwendig, da sie nur mit framework-eigenen Datentypen arbeiten. Die Entwicklung eines einzelnen Plugins kann somit vollständig unabhängig von den anderen Teilen des Frameworks sowie von den zukünftigen Messszenarien geschehen.
Das Resultat ist eine klare Zweiteilung des GREASE-Frameworks in Funktionen zur Datenaufnahme und zur Datenauswertung wie aus Abbildung 3 ersichtlich wird, die die beschriebene Struktur des GREASE-Frameworks aufzeigt. Die Persistenzebene des gesamten Messszenarios bildet dabei das Bindeglied zwischen Datenaufnahme und -auswertung und ist eine Grundvoraussetzung für den Austausch und die Bearbeitung beliebiger Sensorkonfigurationen. Vorhandene Sensorkonfigurationen können nun einfach durch das Hinzufügen beziehungsweise Entfernen von Komponenten (Plugins) verändert werden, ohne das der Datenfluss des Systems dadurch verändert oder angepasst werden muss. Das heißt, dass Messstände verändert werden können, ohne die Auswerteseite anpassen zu müssen. Prinzipiell ist es auch möglich, einen direkten Datenfluss von den Sensordaten-Aufnahmekomponenten zu den Sensordaten-Auswertekomponenten ohne zwischengeschaltete Persistenzebene zu realisieren, und somit eine Echtzeitverarbeitung der Daten zu erreichen. Dabei handelt es sich jedoch nicht um das vorrangige Einsatzgebiet des GREASE-Frameworks, das in Hinblick auf automatisierte Langzeitdatenerfassung und ressourcenbeschränkte Systeme ohne Nutzerinteraktion entwickelt wurde.

4 Implementierung

Das Framework wurde in Java implementiert, um eine hohe Plattformunabhängigkeit zu gewährleisten. Prinzipiell können auch Komponenten unterschiedlicher Programmiersprachen zusammenarbeiten, wenn der verwendete Controller dies unterstützt. Dazu wurde eine zentrale Kernbibliothek[7] geschaffen, die die gesamte Framework-Logik enthält und zur Modulentwicklung genutzt werden kann. Die Kommunikation der PluginController mit den jeweiligen Modulen erfolgt dabei über definierte Interfaces, die so allgemein und generisch wie möglich gestaltet wurden. Die erhöhte Flexibilität vereinfacht die Einbindung von Drittanbieter-Modulen und gewährleistet eine Abwärtskompatibilität im Zuge von Weiterentwicklungen [Bro96]. Konfigurationen und Einstellungen werden im XML-Format übermittelt. Dies begünstigt eine spätere Erweiterbarkeit für anwendungsspezifische Problemstellungen.

4.1 Ablauf

Beim Start des Controllers lädt dieser eine Konfigurationsdatei, die Informationen zum aktuellen Projekt sowie zur Struktur der SensorReader-Komponenten enthält. Der Controller initialisiert daraufhin alle genutzten Plugins und startet die Datenaufnahme. Die einzelnen SensorReader-Module arbeiten dabei threadbasiert und somit (quasi-) parallel. Mit dem Empfang eines Datensatzes an einem SensorReader wird von diesem ein SensorData-Objekt erzeugt. Ein solches SensorData-Objekt besteht aus einem beschreibenden Namen (Tag), wie beispielsweise *gps* für Positionsdaten, und einer Liste von SensorValue-Objekten. Ein SensorValue-Objekt besteht wiederum aus der Beschreibung des Datentyps des Wertes, dem konkreten Datenwert und einem oder mehreren Attributen, die zusätzliche Informationen zu dem aufgenommenen Wert liefern können, wie die physikalische Einheit. In der bei Veröffentlichung dieses Papers aktuellen Version[8] werden die in Tabelle 1 aufgelisteten Datentypen vom Framework unterstützt.

Tabelle 1: Vom GREASE-Framework unterstützte Datentypen

Typ	**Bedeutung**
INT	Einfache Ganzzahl
FLOAT	Gleitkommazahl
DATE	Ein Datumswert
TIMESTAMP	Ein Zeitstempel (Datum + Uhrzeit)
STRING	Eine Zeichenkette
IMAGE	Bilddaten

Das erstellte Objekt wird nun dem Controller gemeldet, der es an eventuelle Filter-Plugins

[7] Java-Bibliothek: GREASE.Core.jar
[8] GREASE.Core Version 1.3

weiterleitet, in Relation zu den Daten anderer SensorReader setzt und mit diesem zu einem Datensatz zusammenfasst. Diesen gibt er schließlich an das Persistenzplugin zur Speicherung weiter. Da der Controller keine Kenntnis über die Art der Daten hat, sprich über deren Bedeutung, ist er anwendungsunabhängig und kann im Gegensatz zu Applikationen wie jBEAM universell eingesetzt werden. Da jBEAM den Ansatz von Projektdateien verfolgt, ist immer eine Anpassung des aktuellen Projektes an das Messszenario notwendig, da Datenaufnahme und –auswertung in diesem Fall voneinander abhängig sind. Das mitgelieferten Persistenz-Plugin des Frameworks übertragt die Daten in ein definiertes XML-Format, jedoch kann der Anwender dieses durch ein eigenes Plugin ersetzen, dass ein beliebiges anderes Format oder Persistenzverfahren verwendet. Darüber hinaus unterstützt das gesamte Framework die Lokalisierung von Inhalten und bietet den Plugins und dem Anwender entsprechende Schnittstellen zur Einstellung von Sprache und Region. Alle bisher entwickelten Plugins besitzen sowohl Unterstützung für die englische als auch die deutsche Sprache. Im einzelnen stehen mit der aktuellen Version des GREASE-Frameworks die in Tabelle 2 gezeigten Plugins zur Verfügung.

Tabelle 2: Mit dem Framework ausgelieferte Standard-Plugins

SensorReader-Plugins	
MTS310Reader	Auslesen von Messdaten der Crossbow MTS310 Sensorboards
GpsReader	Aufnahme von Positionsdaten über NMEA 0183 Standard
ObdReader	Auslesen von Informationen über die OBD-Schnittstelle von Fahrzeugen
CamImageReader	Aufnahme von Bilddaten über bildgebende Geräte wie Webcams
UdpSensorReader	Empfang von Messdaten über UDP
DataFilter-Plugins	
LiveDataViewer	Zeigt den Datenfluss innerhalb des Frameworks
NetVis	Visualisiert Aktivität innerhalb eines Sensornetzes
Persistenz-Plugins	
XML-LogWriter	Speichert Datensätze im XML-Format
XML-LogReader	Liest Messdaten aus dem XML-Format
DB-LogReader	Liest Messdaten aus einer Archivdatenbank
LogReader-Plugins	
LogFuse	Verschmelzung mehrerer SensorLogs
CSVOutput	Speicherung der Messdaten im CSV-Format
ImageExtractor	Auslesen der aufgenommenen Bildaten eines Logs
DBOutput	Archivierung der Daten in einer Datenbank
GEarthOutput	Darstellung der Messdaten in Google Earth

5 Anwendung

Das vorgestellte Framework wurde im Zuge mehrerer Sensornetz-Szenarien an der TU Chemnitz entwickelt. Im Folgenden soll deshalb der Einsatz anhand eines konkreten Anwendungsbeispiels aus dem Bereich Automotive Sensoring and Monitoring erläutert werden [VLCH10]. Im Rahmen mehrerer Langzeitmessungen mit Serienfahrzeugen wurde deren Bordelektronik mit zusätzlicher Sensorik erweitert, die bordnetzunabhängige Messdaten bezüglich Temperatur, Helligkeit, Beschleunigung sowie Magnetfeldstärke zur Verfügung stellt. Zusätzlich wurde ein GPS-Sensor im Wagen verbaut, der kontinuierlich Positionsdaten, Geschwindigkeitsangaben sowie aktuelle Höheninformationen des Wagens liefert. Ebenso wird der OBD-Port [9] des Fahrzeugens genutzt, um weitere fahrzeugspezifische Informationen zu erhalten. Mit Hilfe der gewonnenen Sensorinformationen sollen Rückschlüsse auf bestimmte Fahrsituationen geschlossen werden, die anhand der GPS-Daten später verifiziert werden können. Auch Verschleißanalysen stehen im Fokus der Betrachtungen. Durch die Korrelation aller Sensorinformationen mit den fahrzeuginternen Borddaten entstehen auf diese Weise detaillierte Fahrerprofile, welche wiederum der Optimierung der Fahrzeugcharakteristik dienen. Abbildung 4 verdeutlicht schematisch den Aufbau eines solchen Messszenarios. Neben dem GPS-Empfänger und dem OBD-Anschluss befinden sich im Fahrzeug fünf Sensorboards an strategischen Positionen. Alle Informationen fließen in einer gemeinsamen Datenbasis (engl. sink, Datensenke) zusammen. Für das Auslesen der Sensorboards wurde ein herstellerspezifisches SensorReader-Modul implementiert, welches über den seriellen Port ankommende Daten klassifiziert und als abstrahierte Datenobjekte an den Controller übergibt. Gleiches gilt für ein separat implementiertes Modul für den GPS-Sensor, welches alle NMEA 0183[10] standardkonformen Geräte [KH05] unterstützt. Für das Auslesen der Daten über die OBD-Schnittstelle wurde ein weiteres SensorReader-Modul entwickelt, welches die Funktionen einer eigens entwickelten Java-OBD-Bibliothek nutzt. Zur Synchronisation der Daten wird während der Versuchsinitialisierung ein ausgewählter SensorReader als Master definiert. Da der GPS-Sensor neben den reinen Positionsdaten auch ein exaktes Zeitsignal liefert, repräsentiert dieser im vorgestellten Szenario den globalen Taktgeber (Master) des Systems. Anzumerken ist jedoch, dass das Framework nicht gezwungenermaßen nach einem Zeitsignal synchronisiert werden muss. Das heißt der Taktgeber des Systems ist, im Gegensatz zu anderen Lösungen, vom Anwender frei wählbar. Dies ist insbesondere bei Sensoren, die keine eigene Scheduling-Lösung besitzen, von Vorteil, da nicht beispielsweise jede Sekunde gemessen wird, sondern immer im Moment des Empfangs eines Signals dieses Sensors. Somit ist dieser Sensor der Primärschlüssel und alle anderen Datensätze des Sensornetzes werden diesem untergeordnet. Auch an Messständen an denen kein synchronisiertes Zeitsignal zur Verfügung steht, kann damit eine Messung vorgenommen werden, bei der die Messdaten dennoch in Bezug zueinander gesetzt werden können. Die Speicherung der aufgenommenen Daten erfolgt über das standardmäßige XML-Persistenzmodul des GREASE-Frameworks. Ein Auszug aus einer solchen SensorLog-Datei ist in Abbil-

[9] OBD ... On-Board-Diagnosis, Daten über aktuellen Fahrzeugzustand

[10] NMEA 0183 ... National Marine Electronics Association 0183 - Standard für die Kommunikation zwischen Navigationsgeräten untereinander und mit Computern

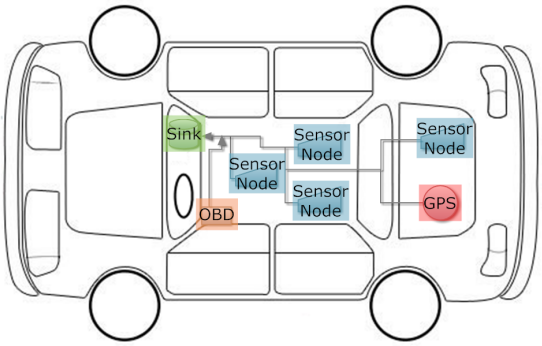


Abbildung 4: Schematische Darstellung des Sensornetzes im Fahrzeug

dung 5 dargestellt.

Für die anschließende Auswertung der ermittelten Daten wurden zwei Auswertekomponenten geschaffen. Dabei handelt es sich einerseits um ein Exportmodul, welches die Sensorwerte für die Einspeisung in die zentrale Datenbank vorbereitet und anschließend überträgt und andererseits um eine zweite Komponente, die die Messdaten für Google Earth aufbereitet. Dabei dienen die in die Datenbank übertragenen Datensätze nicht nur der Archivierung, sondern können ihrerseits wieder in den Auswerteprozess geführt werden und ebenso durch Auswertemodule verarbeitet werden, wie die Daten aus den Log-Dateien. Dabei müssen die Auswertemodule nicht angepasst werden, da der entsprechende SensorLogReader aus der Datenquelle Framework-Objekte erzeugt, mit denen die Auswerteplugins arbeiten. Der umgesetzte Gesamtprozess zur Datenerfassung, -verarbeitung und -aufbereitung ist in Abbildung 6 ersichtlich. Alle geschaffenen Komponenten sind generisch und folglich für eine Vielzahl möglicher Anwendungsszenarien und Hardwarekonfigurationen einsetzbar. Für eine detaillierte Auswertung der Messergebnisse ist auch der Import in jBEAM möglich und sinnvoll. Über ein entwickeltes CSV-Exportmodul[11] lassen sich die Daten problemlos konvertieren.

[11] CSV ... Comma Separated Values, durch Kommata getrennte Datenwerte

```
<?xml version="1.0" encoding="UTF-8" ?>
<sensorlog targetProjectID="12" started="Fri May 25 12:37:12 CEST 2012">
    <dataset id="0">
        <gps friendlyName="GPS-Sensor" synchronized="true">
            <longitude type="FLOAT">12.9274235</longitude>
            <latitude type="FLOAT">50.839626</latitude>
            <speed type="FLOAT" unit="kmh">0.0</speed>
            <timestamp type="TIMESTAMP">Fri May 25 12:37:13 CEST 2012</timestamp>
        </gps>
        <obd friendlyName="OBD-SensorReader">
            <CalculatedEngineLoadValue type="FLOAT" Unit="%">59.21</CalculatedEngineLoadValue>
            <EngineCoolantTemperature type="FLOAT" Unit="°C">82.0</EngineCoolantTemperature>
            <EngineRPM type="FLOAT" Unit="rpm">1704.0</EngineRPM>
            <VehicleSpeed type="FLOAT" Unit="km/h">35.0</VehicleSpeed>
            <IntakeAirTemperature type="FLOAT" Unit="°C">36.0</IntakeAirTemperature>
            <ThrottlePosition type="FLOAT" Unit="%">25.88</ThrottlePosition>
            <RuntimeSinceEngineStart type="FLOAT" Unit="s">271.0</RuntimeSinceEngineStart>
            <FuelLevelInput type="FLOAT" Unit="%">8.62</FuelLevelInput>
        </obd>
        <cam friendlyName="CamImageReader">
            <img type="IMAGE" encoding="base64" quality="80"><![CDATA[<BILDDATEN>]]></img>
        </cam>
    </dataset>
    <dataset id="1">
    ...
    </dataset>
...
</sensorlog>
```

Abbildung 5: Auszug aus einer XML-Log-Datei, die Messdaten enthält

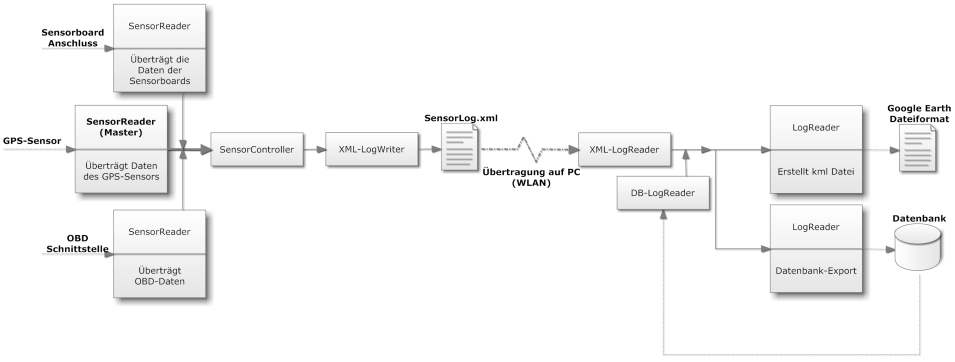
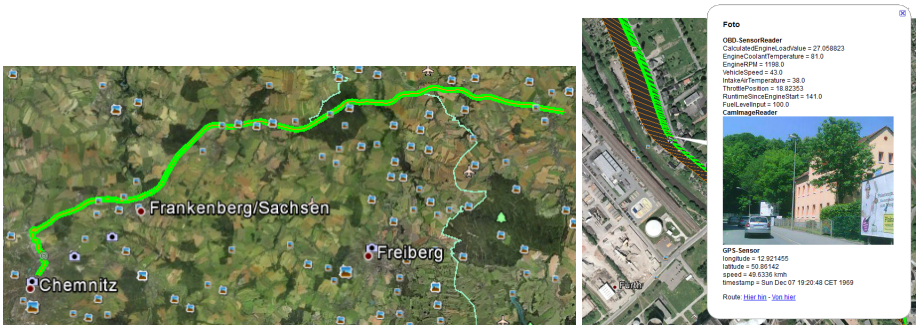


Abbildung 6: Anwendung des Frameworks auf das beschriebene Szenario

6 Auswertung

In den folgenden Abbildungen werden einige zentrale Visualisierungsmöglichkeiten präsentiert, welche die Basis für weitere Analysen bilden. Die konkrete Auswertung und Interpretation der gesammelten Daten ist jedoch nicht Teil dieser Arbeit. Infolgedessen sollen die Abbildungen nur exemplarisch die vielfältigen Möglichkeiten der generischen Ausgabemodule innerhalb des vorgestellten Frameworks verdeutlichen. Das GoogleEarth-Export-Modul ermöglicht die Umwandlung der Messdaten in das GoogleEarth-Datenformat[12], sodass diese in der GoogleEarth-Anwendung betrachtet werden können. Dazu wird eine Route gemäß der GPS-Daten der Log-Datei erstellt (s. Abbildung 7a) und es erfolgt die Erstellung von Routenpunkten für jeden Datensatz der Log-Datei. Die Daten dieser Routenpunkte können dann in Google Earth in einer Detailansicht betrachtet werden (s. Abbildung 7b). Darüber hinaus können Messwertverläufe, wie die Geschwindigkeit entlang der Route als Höhenprofil dargestellt werden, wie in Abbildung 8 gezeigt. Auf Basis der Zeitdaten kann darüber hinaus einerseits die Fahrt in Google Earth mit Hilfe der Zeitleiste animiert werden und andererseits die Route in Abschnitte unterteilt werden, dabei kann die Trennung zum Beispiel erfolgen, wenn sich das Fahrzeug für eine bestimmte Zeitspanne nicht bewegt hat.



(a) Gesamtroute (b) Routenpunkt

Abbildung 7: Darstellung einer Route und eines Routenpunkts in Google Earth

[12] KML bzw. KMZ, wenn Bilddaten eingebunden werden sollen

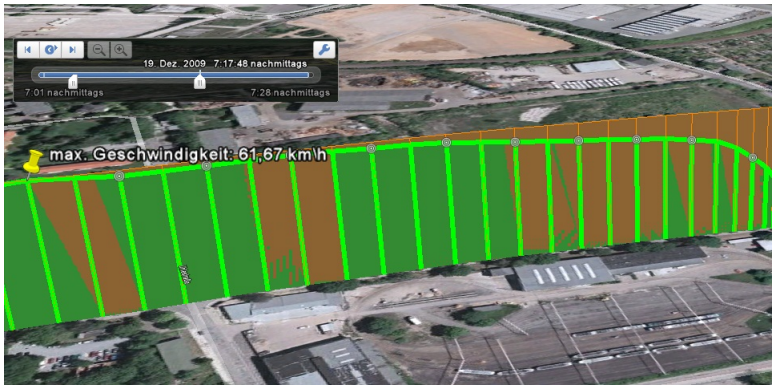
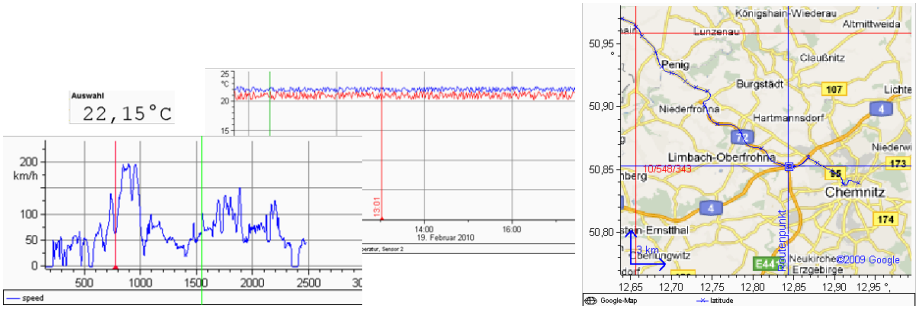


Abbildung 8: Messwertkurve entlang der Routendarstellung in Google Earth

Sowohl über das mitgelieferte CSV-Exportplugin oder ein zu diesem Zweck erstelltes Importplugin für jBEAM kann eine weiterführende Auswertung der Sensordaten auch in diesem Programm erfolgen. Dadurch können alle Visualisierungs- und Statistikfunktionen von jBEAM verwendet werden, ohne dass bei der Datenaufnahme auf bestimmte Einschränkungen bezüglich der jBEAM-Kompatibilität der Daten geachtet werden muss. Klassische Darstellungen wie Messwertkurven sind dabei ebenso möglich wie die Darstellung einer Karte und Anzeige von zusätzlichen Informationen in dieser. Darüber hinaus können verschiedene Elemente auch miteinander synchronisiert werden, sodass für den Nutzer beispielsweise beim Klick in die Karte die entsprechende Stelle in allen Graphen markiert wird. Bei Verwendung des Importplugins besteht ein Vorteil darin, dass die Messdaten einfach gegen die einer anderen Log-Datei ausgetauscht werden können und die Auswertung dann sofort für diese erfolgt, ohne dass das jBEAM-Projekt bearbeitet werden muss.



(a) Verschiedene Sensorwertverläufe (b) Ein Streckenverlauf

Abbildung 9: Visualisierung von Messdaten in jBEAM

7 Zusammenfassung und Ausblick

Die vorgestellte Arbeit beschreibt die Umsetzung einer umfassenden Datenerfassungsumgebung, die eine flexible Basis für weiterführende wissenschaftliche Arbeiten im Bereich der Sensordatenaggregation und -fusion bietet. Im Zuge der Messwertanalyse können nun weiterführende Algorithmen zum Einsatz kommen, welche auf einer gemeinsamen und umfänglichen Datenbasis arbeiten. Somit sind nun im vorgestellten Szenario gezielt Rückschlüsse auf bestimmte Fahrerprofile sowie auf modellspezifische Merkmale der Fahrzeugcharakteristik möglich. Das GREASE-Framework erfüllt hierbei die Anforderungen der Flexibilität, der ressourcensparenden Ausführung und der vollen Automatisierbarkeit, die durch andere Softwarelösungen nicht oder nur teilweise realisiert werden können. Jedoch genau die Kombination dieser drei Punkte macht das geschaffene Framework zu einer Lösung, die es beispielsweise für den vorgestellten Anwendungsfall erst einsetzbar und die Messdatenaufnahme erst möglich macht. Dabei wird es auch für andere Sensordatenprojekte an der Technischen Universität Chemnitz eingesetzt oder noch eingesetzt werden, sodass es sich bei der Messdatenaufnahme um einen standardisierten Prozess handelt und insbesondere die Datenaufbereitung und –archivierung für neue Projekte erheblich vereinfacht wird. Somit kann eine allgemeine Wissensbasis projektübergreifend aufgebaut werden, die einfach, universell und immer gleichartig zugreifbar ist.

Literatur

[Bro96] A. W. Brown. *Component-Based Software Engineering*. John Wiley & Sons, 1996.

[KH05] E. Kaplan und C. Hegarty. *Understanding GPS: Principles and Applications*. Artech House Publishers, 2. Auflage, 2005.

[SRC84] J. Salter, D. Reed und D. Clark. End to End Arguments in System Design. *ACM Transactions on Computer Systems 2*, 2(4):277–288, 1984.

[VLCH10] M. Vodel, M. Lippmann, M. Caspar und W. Hardt. A Capable, High-Level Scheduling Concept for Application-Specific Wireless Sensor Networks. *Proceedings 4th International Symposium on Information Technology*, June 2010.

Chemnitzer Informatik-Berichte

In der Reihe der Chemnitzer Informatik-Berichte sind folgende Berichte erschienen:

CSR-06-07 Karsten Hilbert, Guido Brunnett, A Texture-Based Appearance Preserving Level of Detail Algorithm for Real-time Rendering of High Quality Images, August 2006, Chemnitz

CSR-06-08 David Brunner, Guido Brunnett, Robin Strand, A High-Perpormance Parallel Thinning Approach Using a Non-Cubic Grid Structure, September 2006, Chemnitz

CSR-06-09 El-Ashry, Peter Köchel, Sebastian Schüler, On Models and Solutions for the Allocation of Transportation Resources in Hub-and-Spoke Systems, September 2006, Chemnitz

CSR-06-10 Raphael Kunis, Gudula Rünger, Michael Schwind, Dokumentenmanagement für Verwaltungsvorgänge im E-Government, Oktober 2006, Chemnitz

CSR-06-11 Daniel Beer, Jörg Dümmler, Gudula Rünger, Transformation ereignisgesteuerter Prozeßketten in Workflowbeschreibungen im XPDL-Format, Oktober 2006, Chemnitz

CSR-07-01 David Brunner, Guido Brunnett, High Quality Force Field Approximation in Linear Time and its Application to Skeletonization, April 2007, Chemnitz

CSR-07-02 Torsten Hoefler, Torsten Mehlan, Wolfgang Rehm (Eds.), Kommunikation in Clusterrechnern und Clusterverbundsystemen, Tagungsband zum 2. Workshop, Februar 2007, Chemnitz

CSR-07-03 Matthias Vodel, Mirko Caspar, Wolfram Hardt, Energy-Balanced Cooperative Routing Approach for Radio Standard Spanning Mobile Ad Hoc Networks, Oktober 2007, Chemnitz

CSR-07-04 Matthias Vodel, Mirko Caspar, Wolfram Hardt, A Concept for Radio Standard Spanning Communication in Mobile Ad Hoc Networks, Oktober 2007, Chemnitz

CSR-07-05 Raphael Kunis, Gudula Rünger, RAfEG: Referenz-Systemarchitektur und prototypische Umsetzung - Ausschnitt aus dem Abschlussbericht zum Projekt "Referenzarchitektur für E-Government" (RAfEG) -, Dezember 2007, Chemnitz

CSR-08-01 Johannes Steinmüller, Holger Langner, Marc Ritter, Jens Zeidler (Hrsg.), 15 Jahre Künstliche Intelligenz an der TU Chemnitz, April 2008, Chemnitz

CSR-08-02 Petr Kroha, José Emilio Labra Gayo, Using Semantic Web Technology in Requirements Specifications, November 2008, Chemnitz

CSR-09-01 Amin Coja-Oghlan, Andreas Goerdt, André Lanka, Spectral Partitioning of Random Graphs with Given Expected Degrees - Detailed Version, Januar 2009, Chemnitz

Chemnitzer Informatik-Berichte

CSR-09-02 Enrico Kienel, Guido Brunnett, GPU-Accelerated Contour Extraction on Large Images Using Snakes, Februar 2009, Chemnitz

CSR-09-03 Peter Köchel, Simulation Optimisation: Approaches, Examples, and Experiences, März 2009, Chemnitz

CSR-09-04 Maximilian Eibl, Jens Kürsten, Marc Ritter (Hrsg.), Workshop Audiovisuelle Medien: WAM 2009, Juni 2009, Chemnitz

CSR-09-05 Christian Hörr, Elisabeth Lindinger, Guido Brunnett, Considerations on Technical Sketch Generation from 3D Scanned Cultural Heritage, September 2009, Chemnitz

CSR-09-06 Christian Hörr, Elisabeth Lindinger, Guido Brunnett, New Paradigms for Automated Classification of Pottery, September 2009, Chemnitz

CSR-10-01 Maximilian Eibl, Jens Kürsten, Robert Knauf, Marc Ritter , Workshop Audiovisuelle Medien, Mai 2010, Chemnitz

CSR-10-02 Thomas Reichel, Gudula Rünger, Daniel Steger, Haibin Xu, IT-Unterstützung zur energiesensitiven Produktentwicklung,
Juli 2010, Chemnitz

CSR-10-03 Björn Krellner, Thomas Reichel, Gudula Rünger, Marvin Ferber, Sascha Hunold, Thomas Rauber, Jürgen Berndt, Ingo Nobbers, Transformation monolithischer Business-Softwaresysteme in verteilte, workflowbasierte Client-Server-Architekturen, Juli 2010, Chemnitz

CSR-10-04 Björn Krellner, Gudula Rünger, Daniel Steger, Anforderungen an ein Datenmodell für energiesensitive Prozessketten von Powertrain-Komponenten, Juli 2010, Chemnitz

CSR-11-01 David Brunner, Guido Brunnett, Closing feature regions, März 2011, Chemnitz

CSR-11-02 Tom Kühnert, David Brunner, Guido Brunnett, Betrachtungen zur Skelettextraktion umformtechnischer Bauteile, März 2011, Chemnitz

CSR-11-03 Uranchimeg Tudevdagva, Wolfram Hardt, A new evaluation model for eLearning programs, Dezember 2011, Chemnitz

CSR-12-01 Studentensymposium Informatik Chemnitz 2012, Tagungsband zum 1. Studentensymposium Chemnitz vom 4. Juli 2012, Juni 2012, Chemnitz

CSR-12-02 Tom Kühnert, Stephan Rusdorf, Guido Brunnett, Technischer Bericht zum virtuellen 3D-Stiefeldesign, Juli 2012, Chemnitz

CSR-12-03 René Bergelt, Matthias Vodel, Wolfram Hardt, Generische Datenerfassung und Aufbereitung im Kontext verteilter, heterogener Sensor-Aktor-Systeme, August 2012, Chemnitz

Chemnitzer Informatik-Berichte

ISSN 0947-5125

Herausgeber: Fakultät für Informatik, TU Chemnitz
Straße der Nationen 62, D-09111 Chemnitz