DIPLOMARBEIT

ENTWURF UND PROTOTYPISCHE IMPLEMENTIERUNG EINER FLEXIBLEN SOTIS ARCHITEKTUR

CHRISTOPHER SCHAUMANN

Technische Universität Dortmund
Fachbereich Informatik
Lehrstuhl XII

30.09.2009

Gutachter
Prof. Dr.-Ing. Olaf Spinczyk
Prof. Dr. Gerrit Kalkbrenner

Für meine Freundin & meine Eltern

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1

Einleitung

1.1 Motivation

Deutschlands Straßen sind voll. Laut den Daten des Statistischen Bundesamtes Deutschlands [Des08] teilten sich Anfang 2007 48.989.000 Kraftfahrzeuge, darunter 41.019.700 Personenkraftwagen, rund 231.400 km überörtliches Straßennetz[1]. Würden sich alle Personenkraftwagen gleichzeitig im Straßennetz aufhalten, so würden sich durchschnittlich 177 Personenkraftwagen auf einem Kilometer befinden.

Im Jahre 2006 zählte der Allgemeine Deutsche Automobil-Club (ADAC) in zwölf Ferienwochenenden von Ende Juni bis Mitte September 967 Staus mit einer Länge von zehn Kilometern und mehr [Spi06]. Aneinandergereiht würde eine 13.852 km lange Autoschlange entstehen, die etwa der Strecke von Deutschland nach Australien entspricht. Bereits im Jahre 1999 berichtete der ADAC [Spi99], dass der tägliche Stau auf Deutschlands Autobahnen, bedingt durch Zeit und Spritverlust, einen jährlichen finanziellen Schaden von umgerechnet 7,7 Milliarden Euro verursacht. Ursachen für Staus auf Autobahnen sind Baustellen, Pannen und Unfälle. Letztere nahmen in den vergangenen Jahren erfreulicher Weise permanent ab, aber dennoch belief sich die Anzahl von polizeilich erfassten Autobahnunfällen für 2007 auf insgesamt 20.466 [Rad08].

Heutzutage werden Autobahnen und hoch frequentierte Straßen zentral von Verkehrsleitstellen überwacht. Beim Auftreten von Gefahren[2] werden entsprechende Informationen per Rundfunk oder auch als TMC Meldung an die Verkehrsteilnehmer übermittelt [BMV08a]. Erfasst werden diese Informationen über eine Vielzahl von unterschiedlichen Sensoren[3] entlang der Strecken. Bis vorliegende Rohdaten gezielt als Information weitergeleitet werden können, sind Verzögerungen von einer halben Stunde

[1] Überörtliches Straßennetz: Autobahnen, Bundesstraßen, Landstraßen und Kreisstraßen
[2] z. B.: Unfall, Stau oder gefährliche Wetterveränderungen
[3] z. B. durch Videokameras oder Kontaktsensoren unter der Fahrbahndecke

durchaus möglich [WER+05]. Diese Latenz kann für wichtige Verkehrswarnungen[4] schon zu lang sein. Aus diesem Grund ist die Idee entstanden eine Infrastruktur zu verwenden, die die Aktualität der Informationen besser gewährleistet, indem die zentrale Struktur einer dezentralen weicht: Ein Self-Organizing Traffic Information System (SOTIS) bietet die Möglichkeit Daten dezentral zu sammeln, weiterzuleiten und eigenständig zu agieren. In diesem Sinne ...

Ach, wären Menschen doch so vernünftig wie Fische!
In Fischschwärmen gibt es keinen Stau. [Drö03]

1.2 Ziele der Arbeit

Mit dieser Diplomarbeit soll eine Architektur für ein selbstorganisierendes Verkehrsinformationssystem (engl. Self-Organizing Traffic Information System (SOTIS)) geschaffen werden. In einem ersten Schritt soll berücksichtigt werden, was die Aufgaben von einem selbstorganisierenden Verkehrsinformationssystem sind und was "Architektur" im Kontext der Informatik überhaupt bedeutet. In einem weiteren Schritt soll dann eine Architektur für ein SOTIS passend zu den identifizierten Aufgaben entworfen werden. Der nächste Schritt wird es sein, den entstandenen Entwurf prototypisch umzusetzen. Es soll dabei geklärt werden, wie dieses geschieht, d. h. welche Sprache, Framework, Technologie, etc. dazu verwendet werden. Um zu zeigen, dass die entworfene und prototypisch programmierte Architektur auch funktioniert, soll diese auf einem SOTIS geeigneten Gerät getestet werden.

1.3 Aufbau der Arbeit

Der Aufbau dieser Arbeit ist in Abbildung 1.1 zu sehen und inhaltlich wie folgt gegliedert:

Kapitel 1 vermittelt einen ersten Eindruck über die Thematik und gibt einen kurzen Überblick über den Aufbau der Arbeit.

Kapitel 2 stellt wichtige Projekte im Bereich der Fahrzeug-Kommunikation respektive der mobilen Kommunikation vor und wirft abschließend einen Blick auf ein SOTIS.

Kapitel 3 beschäftigt sich mit der Frage was Architektur im Bereich der Informatik überhaupt ist. Es werden u. a. verschiedene Mittel, Prinzipien und Perspektiven vermittelt.

Kapitel 4 abstrahiert erste Anforderungen an den Entwurf und analysiert den Rahmen in dem die Architektur umgesetzt werden soll.

[4] wie z. B. das Ende eines Staus in einer Kurve

Kapitel 5 beschreibt den Entwurf einer Architektur auf Basis der in Kapitel 2 bis 4 gewonnenen Kenntnissen.

Kapitel 6 dokumentiert die in Kapitel 5 entworfene Architektur und gibt einen detaillierten Blick auf die Komponenten des Systems.

Kapitel 7 evaluiert die entworfene Architektur.

Kapitel 8 schließt diese Arbeit ab und stellt die wichtigsten Punkte noch einmal heraus.

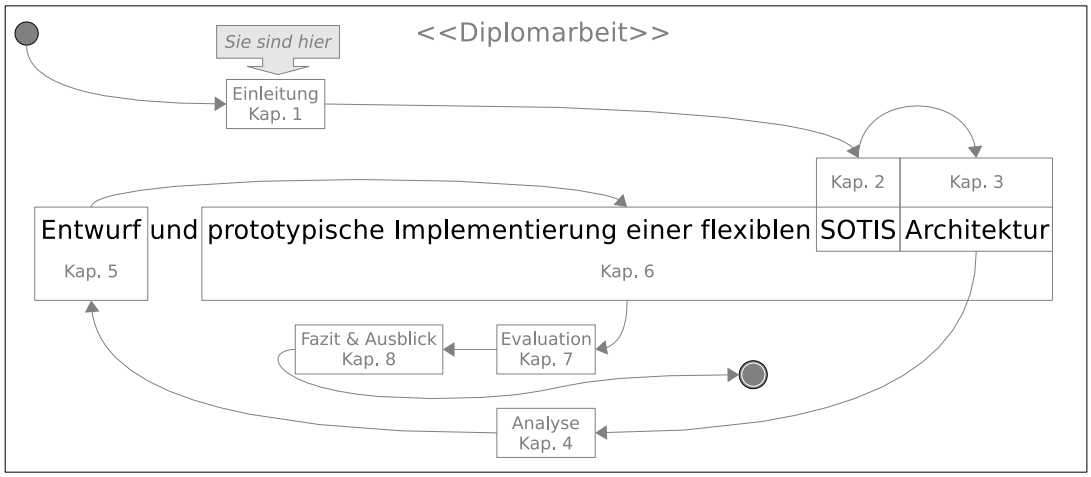

Abbildung 1.1: Aufbau der Diplomarbeit

Kapitel 2

Fahrzeugverkehr

Man kann nicht nicht kommunizieren! [Paul Watzlawick]

2.1 Nagel-Schreckenberg-Modell

Das Nagel-Schreckenberg-Modell von Kai Nagel und Michael Schreckenberg ist ein einfaches Modell zur Simulation von Straßenverkehr [Nag95, NS92]. Es liefert Aussagen über die Entwicklung des Straßenverkehrs bezogen auf die Verkehrsdichte (Fahrzeug pro Streckenabschnitt) und soll einen ersten Eindruck vermitteln, wie Fahrzeugverkehr durch die Wahl eines formalen Weges beschrieben werden kann.

2.1.1 Modell Beschreibung

Kernstück des Modells ist ein eindimensionales Array mit einer endlichen Größe L. L repräsentiert einen Straßenabschnitt, wobei jeder Eintrag in dem Array entweder genau ein Fahrzeug enthalten kann oder leer ist. Jedes Fahrzeug hat eine bestimmte Geschwindigkeit[1] $v \in [0..v_{max}]$ die als Integerwert in L gespeichert wird. Die Anzahl von leeren Einträgen vor einem Fahrzeug wird in *gap* festgehalten. Um in dem Modell die zeitliche Komponente zu berücksichtigen und ein realitätsnahes Verhalten der Fahrzeuge zu simulieren, werden die folgenden Schritte nacheinander auf alle Fahrzeuge gleichzeitig angewendet:

1. **Beschleunigung**
   Wenn die Geschwindigkeit v von einem Fahrzeug kleiner ist als die Höchstgeschwindigkeit v_{max} und wenn genug Platz vor dem Fahrzeug ist, d. h. $v \leq gap - 1$, dann wird die Geschwindigkeit von dem Fahrzeug um 1 erhöht.

$$if\ (v \leq gap - 1)\ then\ v := max[v_{max}, v + 1] \tag{2.1}$$

[1] engl. velocity

2. **Reduzierung der Geschwindigkeit**
   Wenn zum vorderen Fahrzeug ein zu geringer Abstand für die aktuelle Geschwindigkeit $v \geq gap + 1$ besteht, dann wird die aktuelle Geschwindigkeit auf den noch verfügbaren Platz gap reduziert. So werden Auffahrunfälle verhindert.

$$if\ (v \geq gap + 1)\ then\ v := gap \tag{2.2}$$

3. **Randomisierung**
   Wird angewendet nach den Schritten 1 und 2. Die aktuelle Geschwindigkeit v von einem Fahrzeug wird mit einer zufällig ausgewürfelten Wahrscheinlichkeit p um 1 verringert. Im langsamsten Fall würde ein Fahrzeug von Geschwindigkeit 1 auf 0 gesetzt werden und somit nicht mehr fahren.

$$with\ probability\ p\ do\ v := max[v-1, 0] \tag{2.3}$$

4. **Fahrzeug bewegen**
   Jedes Fahrzeug wird entsprechend der aktuellen Geschwindigkeit v bewegt.

Das hier vorgestellte Modell mit den vier Schritten beruht auf der Annahme, dass es sich um eine einspurige Straße handelt. Obgleich es ein einfach gehaltenes Modell ist, zeigt es ein nicht-triviales und realistisches Verhalten gegenüber dem realen Fahrzeugverkehr. Dieses wird bei der näheren Betrachtung von Schritt 3, der Randomisierung deutlich, denn dort lassen sich folgende Verhaltensweisen ableiten:

- **Nicht konstante Maximalgeschwindigkeit**
  Wir nehmen an, ein Fahrzeug fährt mit seiner maximalen Geschwindigkeit $v = v_{max}$ und hat freie Fahrt voraus, also keine vorausfahrenden Fahrzeuge $gap >> v$. Die Geschwindigkeit wird in den Schritten 1 und 2 nicht weiter beeinflusst, die maximale Geschwindigkeit bleibt erhalten. Erst Schritt 3 reduziert die Geschwindigkeit mit einer Wahrscheinlichkeit von p auf $v_{max} - 1$. In dem nächsten Durchgang wird durch die Beschleunigung in Schritt 1 die Geschwindigkeit wieder auf v_{max} erhöht, bevor sie mit einer Wahrscheinlichkeit von p in Schritt 3 erneut reduziert wird. Demnach bewegt sich ein Fahrzeug mit einer Häufigkeit von $1 - p$ mit der maximalen Geschwindigkeit $v = v_{max}$. Dieses spiegelt die Tatsache wider, dass der Mensch bei dem Steuern eines Fahrzeuges nicht konstant dieselbe Geschwindigkeit fahren kann, da er bspw. kurzfristig von anderen Dingen abgelenkt wird.

- **Verzögerte Beschleunigung**
  Ein Fahrzeug habe die Geschwindigkeit $v = 0$. Schritte 1 und 3 zusammen lassen das Fahrzeug mit einer Wahrscheinlichkeit von $1 - p$ auf die Geschwindigkeit $v = 1$ beschleunigen. In mehreren Durchläufen würde die deterministische Beschleunigungssequenz $0 \to 1 \to 2 \to \ldots \to v_{max}$ durch eine nichtdeterministische Beschleunigungssequenz wie z. B. $0 \to 0 \to 0 \to 1 \to 2 \to 2 \to \ldots \to v_{max}$ ersetzt werden.

Die nichtdeterministische Beschleunigungssequenz deckt sich mit den Beobachtungen des realen Verhaltens von Menschen. Bspw. wird auf einer Autobahn erst ein paar Sekunden gewartet bevor das Fahrzeug beschleunigt wird, obwohl der Vordermann mittlerweile schon weit vor einem ist.

- **Überreaktion beim Bremsen**
  Bei einer Verringerung der Verkehrsgeschwindigkeit, sei es der bremsende Vordermann oder viele voranfahrende Fahrzeuge, neigen menschliche Fahrer dazu die Situation über zu bewerten. Anstatt den vorhandenen Raum bis zum Vordermann optimal auszuschöpfen (gap), wird mit einer Häufigkeit von p zu viel gebremst $gap-1$ ($if \geq 0$).

2.1.2 Beispiel

Abbildung 2.1 zeigt anhand von fünf Fahrzeugen die in Abschnitt 2.1.1 vorgestellten vier Schritte des Nagel-Schreckenberg-Modells. Der Initialzustand ist t_0 und alle Fahrzeuge befinden sich auf einer einspurigen Straße und haben eine vorgegebene Geschwindigkeit (Zahl in dem Kreis über dem Fahrzeug). Die maximale Geschwindigkeit ist auf $v_{max} = 5$ begrenzt und in jedem Abschnitt befindet sich nur ein Fahrzeug. Zum Zeitpunkt t_1 ist Schritt 1, die Beschleunigung der Wagen um jeweils eine Einheit zu sehen, sofern die Höchstgeschwindigkeit v_{max} noch nicht erreicht ist. Zum Zeitpunkt t_2 wird die aktuelle Geschwindigkeit angepasst bzw. abgebremst, damit es zu keinen Kollisionen zwischen den Fahrzeugen kommt. Im nächsten Schritt t_3 kommt es zu dem willkürlichen langsamer Fahren, im Beispiel sind davon zwei Fahrzeuge betroffen. Zum Schluss fahren in t_4 alle Fahrzeuge die in t_1 bis t_3 festgelegte Strecke. Bei mehreren Durchgängen würde t_4 wieder zum Initialzustand werden und der ganze Ablauf von vorne beginnen.

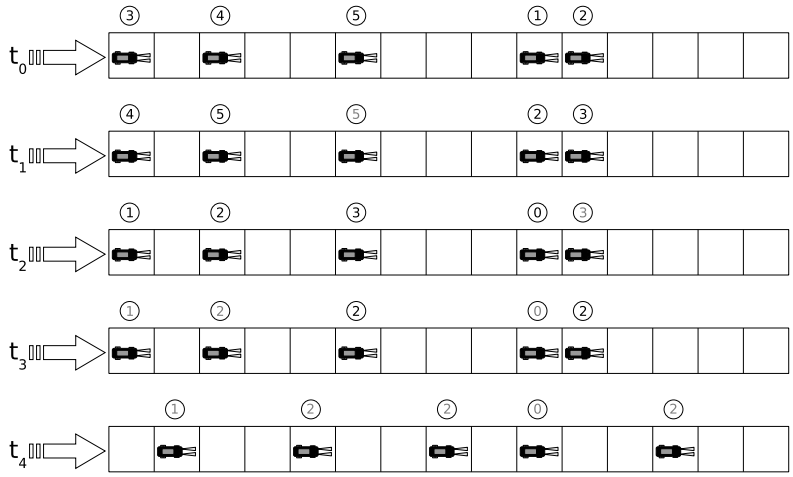

Abbildung 2.1: Beispielablauf des Nagel-Schreckenberg-Modells

Die Abbildungen 2.2 und 2.3 zeigen simulierte Verkehrssituationen in mehreren zeitlichen Durchgängen und bilden somit das Ergebnis mehrerer Durchgänge des in Abbildung 2.1 vorgestellten Durchganges ab. In Abbildung 2.2 ist eine geringe Dichte von $0,03$ Fahrzeugen pro Abschnitt zu sehen. Die aktuelle Geschwindigkeit von Fahrzeugen ist an den Zahlen zu erkennen. Leere Abschnitte sind mit einem . gekennzeichnet. Bei einer Erhöhung der Dichte auf $0,10$ Fahrzeuge pro Abschnitt (vgl. Abbildung 2.3) ist die Entstehung eines Staus an dem vermehrten Auftreten von 0-en zu erkennen. Ferner lässt sich die Auswirkung eines kleinen Staus (zwei 0-en in der obersten Zeile) hin zur Bildung eines größeren Staus (viele 0-en in der untersten Zeile) erkennen sowie eine rückwirkende Fortpflanzung und Verlagerung als Stauwelle nach hinten (bzw. links).

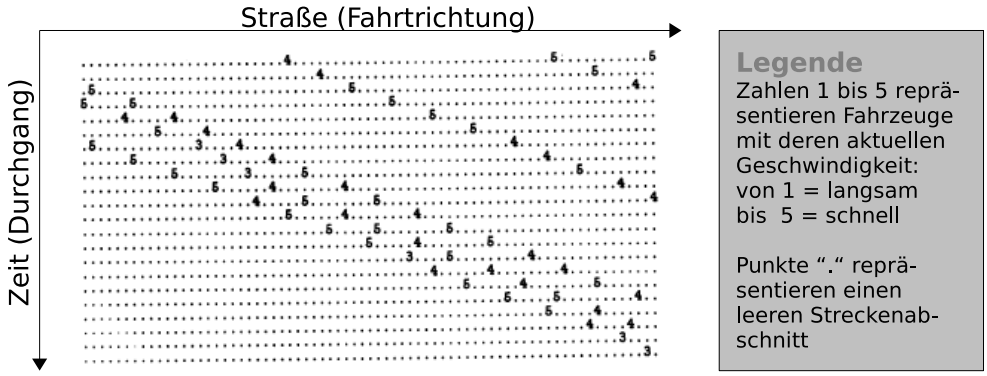

Abbildung 2.2: Simulierte Verkehrssituationen mit Fahrzeugdichte: 0,03 [NS92]

2.2 Erfassung von Verkehrsdaten

Ein Verkehrsinformationssystem benötigt Verkehrsdaten, die die aktuelle Situation beschreiben. Angefangen von der visuellen Beobachtung durch den Menschen, bis hin zur automatisierten Beobachtung, werden in diesem Abschnitt einige unterschiedliche Möglichkeiten der Verkehrserfassung kurz vorgestellt.

2.2.1 Menschliche Erfassung

Durch die visuelle Beobachtung von Verkehr durch Menschen können Verkehrsinformationen erfasst werden. Nach [Kle82] besteht eine Beobachtung aus einem *Oberbegriff*[2],

[2] Die Beobachtung bekommt also eine eindeutige Bezeichnung, einen Titel zugeordnet

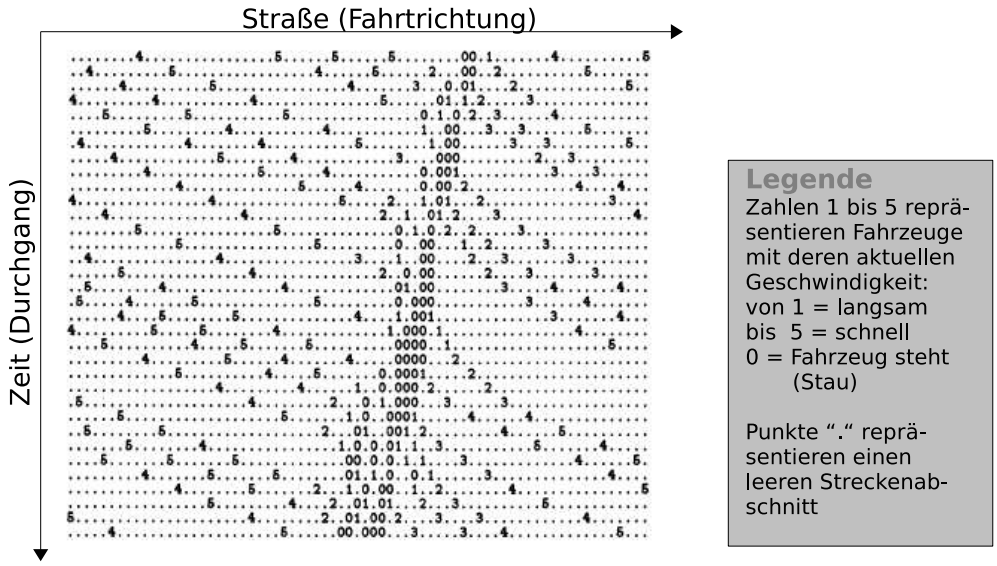

Abbildung 2.3: Simulierte Verkehrssituationen mit Fahrzeugdichte: 0,10 [NS92]

einer *Beschreibung*[3] und einer *Beurteilung*[4] und nennt dazu folgendes Beispiel:

"Das Bremsen mit blockierenden Rädern kann zunächst beschrieben werden durch Ort, Zeitdauer, Intensität des Auftretens; wie es beurteilt wird, hängt ganz von dem zugrundeliegenden Beurteilungskriterium ab: Es kann im Hinblick auf Fahrfertigkeit beim Anfänger als negativ, beim Sportfahrer u. U. als positiv beurteilt werden, wobei sich "negativ" und "positiv" im einen Fall auf die Sicherheit, im anderen Fall auf die geschwindigkeitsbetonte Kurventechnik beziehen."

2.2.2 Maschinelle Erfassung

Durch die ständige voranschreitende Forschung und Entwicklung werden bei der Überwachung des Straßenverkehrs immer mehr technische Geräte eingesetzt. Dieses betrifft sowohl stationäre Geräte, wie z. B. Sensoren an oder auch unter der Fahrbahn, als auch mobile Geräte von Teilnehmern des Straßenverkehrs selbst. Heutzutage enthalten moderne Fahrzeuge eine Vielzahl von Sensoren. Hauptsächlich sind sie für Sicherheit, Komfort und Optimierung des gesamten Motor-Managements zuständig, können aber auch zusätzlich als "Beobachter" der Verkehrssituation verwendet werden. Ein Reihe von Sensoren die in modernen Fahrzeugen eingebaut werden befinden sich im Anhang A (vgl. [Z$^+$07, Bos09]).

[3] Die Beschreibung (= beschreibende Beobachtung) beschreibt die Beobachtung meistens in schriftlicher Form und dient der Tatbestandsaufnahme einer Situation

[4] "Bei der Beurteilung (= beurteilende Beobachtung) wird eine Beziehung zwischen dem Beobachtenden und einem Kriterium hergestellt, wobei das Kriterium *inhaltlich* bestimmt sein muß " [Kle82]

Die von den Sensoren gelieferten Informationen werden u. a. bei Floating Car Data (FCD) Verfahren verwendet, die im Folgenden kurz vorgestellt werden:

Floating Car Data (FCD) ist ein Verfahren bei dem die Position[5] und der Zustand[6] von Fahrzeugen verwendet werden, um Informationen über das aktuelle Verkehrsgeschehen zu erhalten [Tur00]. Die Fahrzeuge sind somit mobile Sensoren des Straßenverkehrs und bieten neben den relativ günstigen Kosten auch aktuelle Daten von weniger befahrenen Straßen, wie bspw. entlegene Landstraßen, wo sich i. d. R. keine Sensoren am Straßenrand befinden. Ausgelegt auf einen dezentralen Ansatz, werden die so gesammelten Daten dann via GSM oder UMTS Mobilfunk an eine Zentrale gesendet, um die aktuelle Verkehrssituation mit den aktuell gewonnenen Erkenntnissen aufzubereiten.

In einer des Deutschen Zentrums für Luft- und Raumfahrt (DLR) im Jahre 2002 durchgeführten Studie wurden insgesamt 300 Berliner City-Funk-Taxis mit entsprechendem Equipment ausgestattet. Die Verkehrsforscher wollten durch die erhaltenen Daten Rückschlüsse auf die Verkehrflüsse in Berlin ziehen, um aktiv Staus entgegenzuwirken [3SA02]. Modifiziert wurde das FCD Verfahren bei dem Projekt FleetNet - Internet on the Road (siehe Abschnitt 2.4.3) und in dezentraler Form verwendet, indem Fahrzeuge direkt miteinander kommunizieren.

Extended Floating Car Data (XFCD) ist ein von BMW weiterentwickeltes FCD Verfahren, das zusätzlich Fahrzeugbetriebsdaten mit berücksichtigt und auf eine dezentrale Sturktur setzt [BMW06]. Zu den Fahrzeugbetriebsdaten zählen u. a. die Schaltzustände von Beleuchtung, Antiblockiersystem (ABS), Automatische Stabilitäts Control (ASC), Außenthermometer, Dynamic Stability Control (DSC), Klimaanlage, Navigationssystem, Bremse, Regensensor, Scheibenwischer und Warnblinker, die zusammen mit aktuellen Geschwindigkeitswerten dazu genutzt werden Ereignis- und Zustandsdaten zu generieren [BMW05]. Tritt bspw. DSC (Dynamische Stabilitätskontrolle) in Verbindung mit einer niedrigen Außentemperatur, erhöhten Scheibenwischerfrequenz und einer niedrigen Fahrgeschwindigkeit auf, kann dieses ein Anzeichen von Eis oder Öl auf der Straße sein.

2.3 Zentrale Verkehrsinformationssysteme

Ein zentrales Verkehrsinformationssystem ist ein System, das Infrastruktur und Funktionen für die Erfassung, Verarbeitung und Verbreitung von Verkehrsinformationen zentral bereitstellt [WER+03]. Abbildung 2.4 skizziert dabei folgende drei Schritte:

[5] durch Verwendung eines GPS Empfängers
[6] z. B. Fahrzeug fährt oder Fahrzeug steht

1. **Erfassung**
   Unterschiedliche Sensoren entlang der Fahrbahn erfassen Daten und leiten diese an ein Verkehrsinformationszentrum (Traffic Information Center (TIC)) weiter. Bspw. können unterhalb der Fahrbahn angebrachte Sensoren den aktuellen Verkehrsfluss messen.

2. **Verarbeitung**
   Im TIC werden alle Verkehrsinformationen zusammengetragen und verarbeitet. Neben den sensorisch erfassten Daten können aber auch von Menschen erfasste Daten, z. B. Polizeimeldungen, mit in die Analyse der bevorstehenden Verkehrssituation einbezogen werden. In unserem Beispiel kann ein durch die Sensoren festgestellter erhöhter Verkehrsfluss als möglicher Indikator für einen anstehenden Verkehrsstau auf dem betroffenen Streckenabschnitt interpretiert werden.

3. **Verbreitung**
   Die im TIC gewonnenen Ergebnisse werden anschließend via Rundfunk (oder Mobilfunknetz) verbreitet und von empfangenden Fahrzeugen individuell ausgewertet.

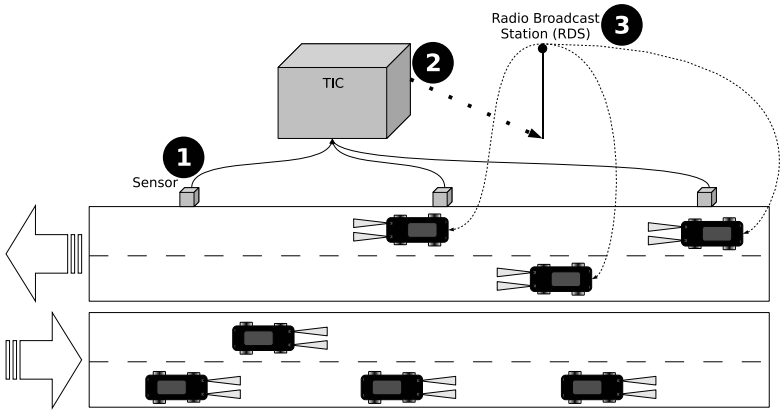

Abbildung 2.4: Zentrales Verkehrsinformationssystem nach [WER+03]

In Deutschland gibt es aktuell zwei zentral organisierte Verkehrsinformationssysteme TMC und TMCpro FM die im Folgenden vorgestellt werden.

2.3.1 Traffic Message Channel (TMC)

Der Traffic Message Channel (TMC) ist ein kostenloser Verkehrsinformationsdienst, der digital über Rundfunksender übertragen wird [BMV08a]. Die Verkehrsinformationen werden dabei u. a. von Detektoren am Straßenrand aber hauptsächlich durch Meldungen von

Polizei und den Staumeldern des ADAC erfasst und an Verkehrsrechnerzentralen weitergegeben. Die Landesmeldestellen[7] werten vorliegende Verkehrsinformationen der Verkehrsrechnerzentralen aus und geben authorisierte TMC-Meldungen an öffentlich-rechtliche Rundfunkanstalten und zum Teil private Rundfunkanbieter zum Aussenden weiter. Übertragen werden dabei die "Angaben zum Ort der Störung" in einem Code aus der Location Code List (LCL), das Ereignis sowie eine zeitliche Gültigkeit der TMC-Meldung [BAS06]. Ein Code ist eine Zahl zwischen 1 und 63487 und beschreibt damit Punkte, Strecken oder auch Gebiete, d. h. es werden keine konkreten Straßennamen, Kreuzungen oder ähnliches übertragen. Laut [NAV08] werden die TMC-Meldungen alle zehn Minuten mit 60 bit/s aktualisiert, was bei einer Fahrzeuggeschwindigkeit von 120 km/h eine Informationsaktualisierung von 20 Minuten bedeutet.

2.3.2 TMCpro FM

Mit TMCpro FM wird ein kostenpflichtiger Verkehrsinformationsdienst bezeichnet der ausschließlich über private Rundfunkanbieter verschlüsselt übertragen wird. Verglichen mit TMC werden die Verkehrsinformationen nicht nur durch Meldungen und Detektoren gesammelt, sondern auch zusätzlich mit wertvollen, anonymisierten Bewegungsdaten von Handys angereichert [Bor09]. Momentan wird das T-Mobile-Mobilfunknetz von namhaften Herstellern wie Becker, Garmin, Mio, LG Electronics, Navigon und Kenwood benutzt, einzig TomTom verwendet das Vodafone-Mobilfunknetz und nennt sein System High Definition Traffic (HDTraffic) [Bor09, Ver07].

2.4 Dezentrale Verkehrsinformationssysteme

Ein dezentrales Verkehrsinformationssystem ist ein System, das Infrastruktur und Funktionen für die Erfassung, Verarbeitung und Verbreitung von Verkehrsinformationen dezentral bereitstellt [WER+03]. Verglichen mit Abschnitt 2.3 bedeutet dieses auf der einen Seite eine wesentlich flexiblere Struktur, da fest verbaute Sensoren nur optional notwendig sind[8], auf der anderen Seite einen höheren Organisationsaufwand der gesamten Informationsverarbeitung[9].

In den vergangenen Jahren wurden zahlreiche Projekte zur Erforschung und kontinuierlichen Verbesserung in dem Bereich der dezentralen Verkehrsinformationssysteme durchgeführt. Abbildung 2.5 skizziert die bekanntesten europäischen Projekte in chronologischer Ordnung. Folgend werden diese und internationale Aktivitäten weiter vorgestellt.

[7] Zentrale Stelle des jeweiligen Bundeslandes

[8] Dieses spart Kosten und ermöglicht zusätzlich den Einsatz in abgelegenen Regionen

[9] Sowohl technisch durch die Auswahl geeigneter Routing-Algorithmen, Hardware, etc. als auch inhaltlich bzgl. adäquater Verarbeitung und Verbreitung der Informationen

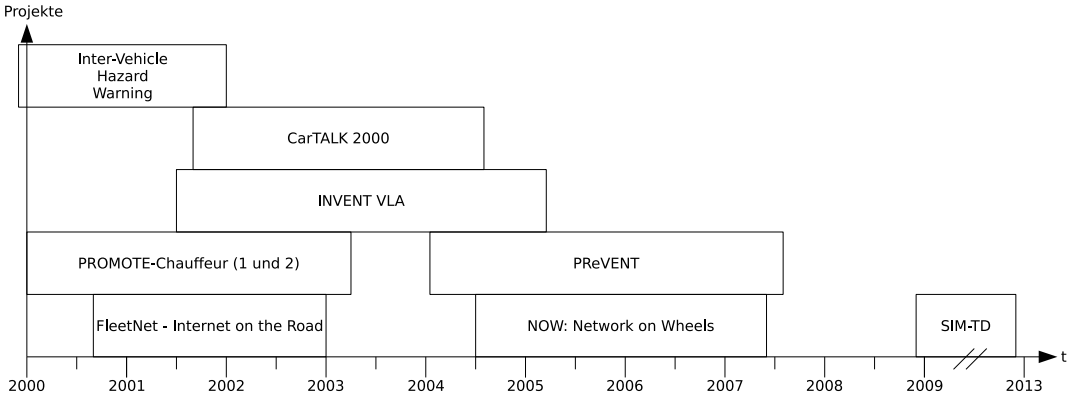

Abbildung 2.5: Projekte dezentraler Verkehrsinformationssysteme

2.4.1 Inter-Vehicle Hazard Warning (IVHW)

Inter-Vehicle Hazard Warning (IVHW) ist ein von der Bundesanstalt für Straßenwesen (BASt) und der Französischen Regierung gefördertes Projekt der Deutsch-Französischen Kooperation (DEUFRAKO), das zwischen 1998 bis 2002 durchgeführt worden ist [DEU03, DEU08, BAS02]. Ziel war die gemeinsame Entwicklung eines Gefahrenwarnsystems auf Autobahnen sowie die Übertragung von Fahrzeug zu Fahrzeug mittels Kommunikationssystem. Neben Algorithmen, die die Relevanz von Warnungen beurteilen, wurden auch Nachrichtenformate für Warnmeldungen und Anzeigen im Fahrzeug festgelegt. Die bekanntesten beteiligten Unternehmen und Institutionen waren ISIS, Renault, Peugeot, Citroën, ESTAR, COFIROUTE, INRETS, DaimlerChrysler und Bosch.

2.4.2 PROMOTE-Chauffeur (I und II)

Die PROMOTE-Chauffeur Projekte I und II unterscheiden sich von den anderen im Bereich der Fahrzeug zu Fahrzeug Kommunikation durchgeführten Projekten dadurch, dass der Fokus hinsichtlich Funktionen und Anwendungen speziell für Lastkraftfahrzeuge ausgerichtet wurden. Im Rahmen von PROMOTE-Chauffeur I wurde die so genannte *Tow-Bar*[10] Anwendung entwickelt und prototypisch in Lastkraftfahrzeugen installiert [Har01]. Zudem wurden Machbarkeits-Untersuchungen für *Platooning*[11] und *Automated Platooning*[12] durchgeführt. In dem Nachfolgeprojekt PROMOTE-Chauffeur II wurde Platooning wieder aufgegriffen und so weiterentwickelt, dass es in drei Lastkraftfahrzeugen integriert und getestet werden konnte. Auf Basis des Tow-Bar Systems aus PROMOTE-Cauffeur I wurde der *CHAUFFEUR Assistant* um verteilte Systemfunktionen erweitert. Dazu gehör-

[10] Tow-Bar bezeichnet die elektronische Kuppelung von zwei Lastkraftfahrzeugen, wobei nur das vordere Fahrzeug einen Fahrer benötigt

[11] Platooning ist wie Tow-Bar mit mehr als zwei Lastkraftfahrzeugen

[12] Automated Platooning ist wie Plattoning ohne Fahrer im ersten Fahrzeug

ten Adaptive Cruise Control (ACC) und Lane Keeping (LK) um bspw. die Bremskontrolle der elektronisch gekoppelten Lastkraftfahrzeuge weiter zu optimieren. Das Projekt hatte eine Laufzeit von Januar 2000 bis Mai 2003. Beteiligt waren u. a. die TUEV Kraftfahrt GmbH, DaimlerChrysler AG, Robert Bosch GmbH, Renault VI und Centro Ricerche Fiat Societa Consortile per Azioni [IST00].

2.4.3 FleetNet - Internet on the Road

In dem Projekt FleetNet - Internet on the Road - war das Ziel, Daten auf Basis eines Adhoc-Funknetzes direkt zwischen Fahrzeugen auszutauschen [Fle02]. In einem ersten Schritt wurde geeignete Funk-Hardware auf die Anforderungen[13] zur Verwendung in diesem Bereich genauer untersucht. Zur Auswahl standen UTRA-TDD, ein Radarsystem und IEEE 802.11. Letzteres wurde ausgewählt und für die Implementierung in einem Demonstrator verwendet. Ein weiterer Schritt war die Entwicklung und Verwendung von effizienten Routingprotokollen, die der dynamischen Struktur eines mobilen Adhoc-Funknetzes gerecht werden kann. Unter der Voraussetzung, dass jedes Fahrzeug über einen GPS-Empfänger verfügt, wurde als Ansatz das positionsbasierte Routing gewählt[14]. Später sollte noch das topologische Routing weiter betrachtet werden. Als letztes wurden Anwendungen und Dienste aus den Bereichen *Kooperative Fahrerassistenzsysteme*, *dezentrale Floating Car Data Application* und *Kommunikations- und Informationsdienste* entwickelt, um die Gesamtfunktionalität des Demonstrators zu zeigen. Dabei stellte sich heraus, dass die unterschiedlichen Anwendungen und Dienste, unterschiedliche Anforderungen an das System bzgl. Bandbreite, Zuverlässigkeit, Reichweite, Positionsgenauigkeit und Verzögerung hatten.

Das vom Bundesministerium für Bildung und Forschung (BMBF) geförderte Projekt lief von September 2000 bis Dezember 2003. Neben den (technischen) Universitäten Braunschweig, Hamburg-Harburg, Hannover und Mannheim waren u. a. DaimlerChrysler AG, Robert Bosch GmbH, Siemens AG und NEC Europe Ltd. als Partner beteiligt.

2.4.4 INVENT - VLA

Intelligenter Verkehr und nutzergerechte Technik (INVENT) ist ein vom BMBF gefördertes Projekt, das zwischen Juni 2001 und Mai 2005 von namenhaften Firmen wie der BMW Group, DaimlerChrysler AG, MAN Nutzfahrzeuge AG, Robert Bosch GmbH und Volkswagen AG durchgeführt worden ist [Bar02]. Festgelegtes Ziel des Projektes war es durch innovative Technologien dem ständig steigenden Verkehrsaufkommen und den dadurch möglicherweise resultierenden Staus und Unfällen aktiv entgegen zu wirken, um die Verkehrssicherheit weiter zu erhöhen. Unterteilt wurde das Projekt in folgende Teilprojekte:

[13] Dazu gehören u. a.: Übertragungsreichweite min. 500 m, Übertragungsrate min. 1 Mbit/s sowie ein lizenzfreies Frequenzband

[14] basierend auf den Arbeiten von [MWH01, M+00]

- **Fahrerassistenz, aktive Sicherheit**
  Fahrumgebungserfassung und Interpretation (FUE), Vorausschauende Aktive Sicherheit (VAS), Stauassistenz (STA), Fahrerverhalten und Mensch-Maschine-Interaktion (FVM), Verkehrliche Wirkung, Rechtsfragen und Akzeptanz (VRA)

- **Verkehrsmanagement 2010**
  Verkehrsleistungsassistenz (VLA), Netzausgleich und Individualverkehr (NIV)

- **Verkehrsmanagement in Transport und Logistik (VMTL)**

Es wurde gezielt nach Antworten auf die Fragen “Wie lässt sich ein Stau schnell auflösen?”, “Wie können Stop-and-Go Wellen gedämpft werden?” und “Wie können hohe Verkehrsflüsse bei dem Einfädeln anderer Fahrzeuge erhalten bleiben?” gesucht, um die Ergebnisse in einem VLA zu berücksichtigen.

2.4.5 CarTALK 2000

CarTALK 2000 ist ein von der europäischen Union gefördertes Projekt, das im August 2001 startete und auf eine Laufzeit von drei Jahren begrenzt war [RMM+02]. Forschungsschwerpunkte des Projekts waren zum einen die Entwicklung von unterschiedlichen Anwendungen aus den Bereichen *Informations- und Warnfunktionen*[15], *Kommunikationsbasierte Steuerfunktionen*[16] und *Kooperative Assistenzsysteme*[17]. Zum anderen sollten für die genannten Bereiche auch geeignete Kommunikationsprotokolle entwickelt und getestet werden. Die Projekt-Koordination übernahm die DaimlerChrysler AG und entwickelte einen Prototypen zusammen mit der Robert Bosch GmbH, Siemens ICN und der Universität Stuttgart. Weitere Partner waren u. a. Centro Ricerce Fiat (CRF) und die Universität Köln.

2.4.6 Preventive and Active Safety Applications (PReVENT)

Das PReVENT Projekt ist Teil des von der EU im sechsten Rahmenprogramm geförderten, globalen Ansatzes für sicheren Straßenverkehr, der von der Automobil- und Zulieferindustrie in Europa eingeschlagen worden ist [SMI+08]. Entwickelt werden sollten präventive Sicherheitsanwendungen in Form eines Assistenten, der den Fahrer abhängig von der Signifikanz und zeitlichen Einschätzung der Situation, erst über eine mögliche Gefahr informiert, bei nicht reagieren des Fahrers erneut warnt, und in einem

[15] z. B. vor einem anderen liegengebliebenen Fahrzeug waren
[16] z. B. wird ein Fahrzeug abgebremst, weil sein (nicht sichtbarer, durch einen LKW verdeckter) Vordermann auch bremst
[17] z. B. Unterstützung bei dem Wechseln von der Beschleunigungsspur auf die Autobahn

letzten Schritt (bei Missachtung der Information und Warnung) aktiv ins Fahrgeschehen eingreift und handelt. Zusätzlich soll der Assistent den Fahrer noch mit weiteren präventiven Sicherheitsanwendungen unterstützen, wie z. B. das Überholen in kritischen Situationen vermeiden, das Fahrzeug automatisch in der aktuellen Fahrspur halten oder den Mindestabstand einzuhalten.

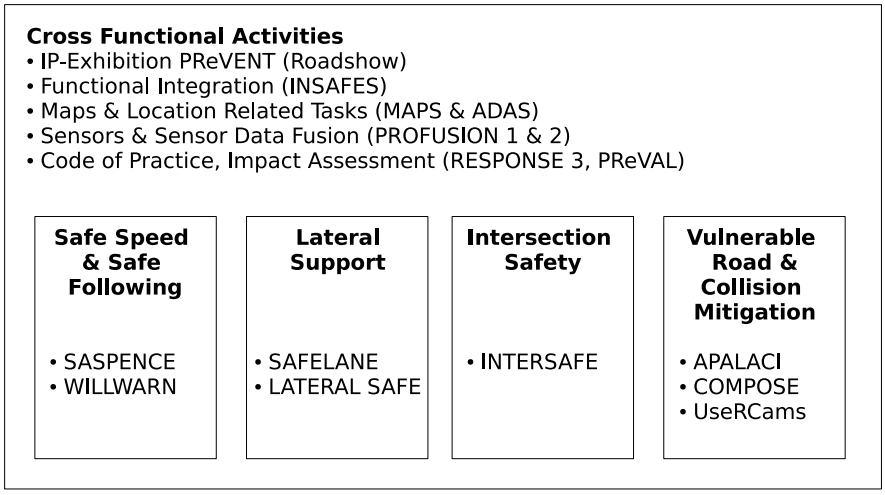

Abbildung 2.6: PReVENT Projektstruktur [SMI+08]

Die grobe Struktur von PReVENT ist zum einen in die Cross Functional Activities (horizontale Aktivitäten) unterteilt, zum anderen finden sich diese auch in allen Teil-Projekten Safe Speed & Safe Following, Lateral Support, Intersection Safety und Vulnarable Road Users & Collision Mitigation wieder, auf die im Detail nicht weiter eingegangen wird. Abbildung 2.6 zeigt die Struktur des gesamten PReVENT Projektes. Einzig das Teil-Pojekt WILLWARN sollte nicht unerwähnt bleiben, da hier der Fokus auf die Fahrzeug zu Fahrzeug Kommunikation gelegt worden ist. Untersucht wurden verschiedene Szenarien wie z. B. das Feststellen und Warnen vor Hindernissen, die sich nicht sichtbar hinter einer Kurve befinden. Hindernisse können dabei gefrorene Straßen, liegengebliebene Fahrzeuge oder auch Baugerüste sein.

PReVENT startete Anfang 2004 und zählt mit ca. 50 beteiligten Firmen und Institutionen aus Industrie, Wirtschaft und Forschung mit zu den größten in diesem Bereich durchgeführten Projekten in Europa.

2.4.7 Network on Wheels (NOW)

Das Ziel des vom BMBF geförderten Projektes Network on Wheels (NOW) war die Spezifikation eines Kommunikationssystems zur Übertragung von Sensordaten und allgemei-

nen Informationen in Fahrzeug-Adhoc-Netzen [Plu05]. Ferner sollten in der Projektlaufzeit von Juni 2004 bis Mai 2008 positionsbasierte Routing- und Forwardingprotokolle für Adhoc-Netze spezifiziert und existierende Kommunikationsprotokolle an die realen Funkbedingungen angepasst werden. Die Projektpartner AUDI AG, BMW AG, DaimlerChrysler AG, Fiat, Renault und VW AG und auch gleichzeitig Mitglieder des Car2Car Communication Consortium (C2C-CC) sollten zusammen mit den Auftragnehmern Carmeq GmbH, Universität Karlsruhe, Universität Mannheim und der TU München die Ergebnisse europaweit standardisieren - unabhängig von Marke und Hersteller. Ein weiterer technischer Schwerpunkt sollte die Entwicklung von Deployment- und aktiven Sicherheitsanwendungen sein, damit die Fahrzeuge z. B. in der Lage sind über einen am Straßenrand befindlichen Hotspot Internet-Anwendungen auszuführen. Neben solchen Anwendungen wurde zusätzlich noch Wert auf die Datensicherheit der Kommunikation in mobilen Fahrzeug Adhoc-Netzen gelegt. Abgeschlossen werden sollte das Projekt mit der Implementierung eines Referenzsystems, konform mit der Standardisierung.

2.4.8 Sichere, intelligente Mobilität - Testfeld Deutschland (SIM-TD)

SIM-TD ist ein von Bundesministerium für Verkehr, Bau und Stadtentwicklung (BMVBS), Bundesministerium für Wirtschaft und Technologie (BMWi) und BMBF gefördertes Projekt mit dem Ziel die Anwendungsmöglichkeiten der Car2Car Communication (C2CC) und Car-to-Infrastructure (C2I) in einem Feldversuch zu untersuchen (vgl. [BMV08b]). Es soll versucht werden die Straßensicherheit durch intelligente Kommunikation zu erhöhen sowie Staus zu reduzieren. Beteiligte Unternehmen sind Audi, BMW, Daimler, Ford, Opel und Volkswagen, Bosch, Continental Teves und Deutsche Telekom. Das Projekt startete im November 2008 und hat eine Laufzeit von vier Jahren.

2.4.9 Projekte in den USA und Japan

Im Jahr 2002 beschäftigte sich, das im Rahmen des Vehicle Safety Communications (VSC) Projektes gegründete VSC Consortium (VSCC), zu dem die sieben in den USA vertretenen Automobilherstellerfirmen BMW, DaimlerChrysler, Ford, GM, Nissan, Toyota und VW gehören, mit Fragen aus dem Bereich der Fahrzeugkommunikation zur Erhöhung der Verkehrssicherheit und Verbesserung von Verkehrsflüssen [SD07]. Die amerikanische Regulierungsbehörde (Federal Communication Commission (FCC)) stellte dazu speziell ein 5.9 Ghz-Frequenzband bereit, um die Dedicated Short Range Communications (DSRC) näher zu untersuchen. In enger Zusammenarbeit mit dem United States Department of Transportation (USDOT) wurden so aus einem großen Fragenkatalog die wichtigsten Funktionalitäten eines DSRC Systems herausgearbeitet:

- Aus dem Bereich Kommunikation zwischen Fahrzeug (On-board Unit (OBU)) und Infrastruktur (Road-side Unit (RSU)):

  - Curve Speed Warning
  - Left Turn Assistant
  - Stop Sign Movement Assistance
  - Traffic Signal Violation Warning

- Aus dem Bereich Kommunikation zwischen Fahrzeug und Fahrzeug:
  - Cooperative Forward Collision Warning
  - Emergency Electronic Brake Lights
  - Lane Change Warning
  - Pre-Crash Sensing

In Japan beschäftigt sich die Advanced Cruise-Assist Highway System Research Association (AHSRA) seit 1996 mit dem Advanced Cruise-Assist Highway System (AHS) um die Erhöhung der Verkehrssicherheit [AHS09]. Wie auch in anderen Projekten bereits kennengelernt, werden durch die Positionierung von unterschiedlichen Sensoren RSU am Straßenrand Verkehrsinformationen gesammelt und ausgewertet. An die OBU übermittelt liefern sie dem Fahrer genaue Verkehrsinformationen. Durch Verwendung einer RSU, in einer für einen Fahrer nicht einsehbaren Kurve, konnte durch den Einsatz der neuen Technik mittels frühzeitiger Vorwarnung, eine Reduzierung der Auffahrunfälle um 79% nachgewiesen werden.

2.5 Diskussion

Die in den Abschnitten 2.3 und 2.4 vorgestellten zentralen und dezentralen Verkehrsinformationssysteme haben alle das gemeinsame Ziel, den Straßenverkehr *positiv* zu beeinflussen und zu verbessern. Während die zentralen Verkehrsinformationssysteme den Vorteil haben, teilweise schon seit Jahren auf dem Markt etabliert zu sein, können die dezentralen Verkehrsinformationssysteme erst in der Zukunft - dank Dedicated Short Range Communications (DSRC) - mit einem breit gestreuten Angebot an Funktionen nachziehen: Sei es ein Fahrzeug, das ein anderes Fahrzeug automatisch per Funktechnik vor sensorisch erfasstem Glatteis in einer Kurve warnt oder ein Lastkraftwagen der autonom einem voranfahrenden Lastkraftwagen folgt.

Neben den Kosten für die Infrastruktur zur Erhebung von Verkehrsdaten durch Personen (TMC) oder Sensoren (TMCpro FM) und den speziell im Fahrzeug verbauten Komponenten und Endgeräten selbst (dezentrale Verkehrsinformationssysteme), ist ein zumindest lokal flächendeckendes Netz zur Kommunikation notwendig. Aber selbst bei ausreichend gegebener Netzabdeckung und vernachlässigbaren Kosten für die Infrastruktur besteht eine weitere Herausforderung in der Standardisierung. Zwar spiegelt Tabelle 2.1 zusammenfassend eine rege Beteiligung von Fahrzeugherstellern an Projekten wieder, ein von

Eigenschaften des Verkehrsinformationssystems	**Zentral**		**Dezentral**							
	TMC	TMCpro FM	IVHW	FleetNet	CarTalk2000	SIM-TD	PROMOTE Chauffeur	INVENT-VLA	NOW	PReVENT (WILLWARN)
Abgeschlossene Markteinführung	●	●								
Am Straßenrand installierte Sensoren verwendet	●	●		●					●	
Kostenpflichtige Nutzung		●								
Schnelle Aktualisierung und Verbreitung von Daten		●	●	●	●	●	●	●	●	●
IEEE 802.11 Funktechnik				●	●		○	○	○	○
Anzahl beteiligter Automobilhersteller (* inkl. LKW Herstellern)			4	1	2	6	5*	4*	6	2
Offene Architektur										

● = trifft zu ○ = trifft sehr wahrscheinlich zu

Tabelle 2.1: Diskussion: Verkehrsinformationssysteme

allen Herstellern akzeptierter Standard wurde bisher unter wirtschaftlichen Aspekten noch nicht umgesetzt.

Eine Alternative zu den vorgestellten Systemen ist ein Self-Organizing Traffic Information System (SOTIS), welches schon bei einer Verbreitung von 1-3% in allen Fahrzeugen Verkehrsinformationen effizient zur Verfügung stellen kann [WER+05]. Ein SOTIS wird in dem nun folgenden Abschnitt vorgestellt.

2.6 Self-Organizing Traffic Information System (SOTIS)

SOTIS steht für Self-Organizing Traffic Information System und ist ein selbst organisierendes Verkehrsinformationssystem das, basierend auf Adhoc Netzwerken für Fahrzeuge (Vehicular Ad Hoc Network (VANET)), Reise und Verkehrsinformationen (Travel and Traffic Information (TTI)) verbreiten kann [WER+03, WER04, WER+05, Wis07]. Dabei kommt es im Gegensatz zu den in Abschnitt 2.3 vorgestellten Verkehrsinformationssystemen ohne zentrale Strukturen aus, da alle Fahrzeuge direkt miteinander kommunizieren (Car2Car Communication (C2CC)). Abbildung 2.7 skizziert die Kommunikation (1) zwischen Fahrzeugen (2).

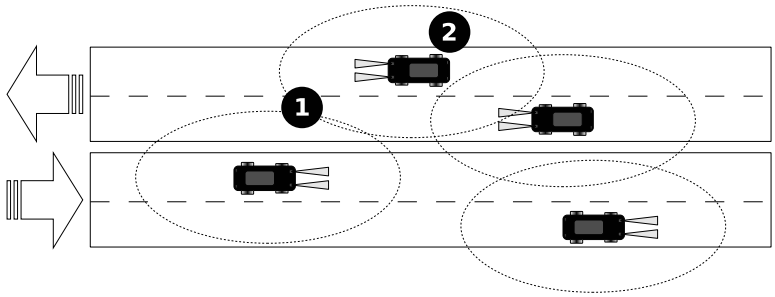

Abbildung 2.7: Dezentrales Verkehrsinformationssystem nach [WER+03]

2.6.1 Komponenten

In einer einfachen Ausführung benötigt ein SOTIS drei Komponenten für einen erfolgreichen Betrieb [WER04]:

- **Wireless Radio Transceiver**
  zum Senden und Empfangen von Signalen zwischen den Fahrzeugen
- **GPS Receiver**
  zur Bestimmung der aktuellen Position eines jeden Fahrzeuges

- **Digitale Karte**
  zur genauen Bestimmung des physikalisch existierenden Ortes

Mittlerweile gibt es einige Handyhersteller, wie z. B. Nokia [Nok09b] oder auch Sony-Ericsson [Son09], die Geräte mit WLAN, GPS Empfänger und Kartenslot (der mit einer Speicherkarte zur persistenten Speicherung digitaler Landkarten verwendet werden kann) anbieten und somit die o. g. Komponenten von Haus aus integriert haben bzw. unterstützen. Dank zusätzlich vorhandenem Bluetooth Chip in Verbindung mit einem Bluetooth Adapter zum Auslesen des Fahrzeug CAN-Busses [CAS06], stehen den potentiellen SOTIS Anwendungen Türen und Tore offen. So sollte es kosteneffizient möglich sein, die unterschiedlichen Sensoren und Messwerte des Fahrzeugs mit in die erfassten Daten des SOTIS einfließen zu lassen. Dieses ist ein nicht zu unterschätzender Faktor für eine erfolgreiche Markteinführung und ausreichende Verbreitung zur Erreichung der 3% Marke (vgl. Abschnitt 2.5).

2.6.2 Datentransfer zwischen SOTIS Teilnehmern

Fahrzeuge die mit einem SOTIS augerüstet sind haben mehrere Möglichkeiten an Verkehrsdaten zu gelangen. Sie können die eigenen vorliegenden Sensordaten zur Beurteilung der aktuellen Verkehrssituation verwenden. Dabei ist die Menge an Daten, die für eine vorausschauende Analyse benötigt werden, durch die Reichweite der Fahrzeugsensoren, abhängig von der zurückgelegten Strecke beschränkt, und stellt somit nur ein unmittelbares Umfeld der aktuellen Verkehrssituation dar. Zusätzlich sollten Fahrzeuge dann noch auf bereits gesammelte und bewertete Informationen von anderen Fahrzeugen oder auch RSU zurückgreifen. Die neuen Fremd-Informationen werden wiederum mit den eigenen kombiniert und erneut weiteren Fahrzeugen zur Verfügung gestellt. Für den Transfer dieser Daten wird der so genannte “Segment-Oriented Data Abstraction and Dissemination (SODAD) Algorithmus” verwendet, der in die folgenden zwei Teilschritte aufgeteilt werden kann.

1. Schritt: Segment-Oriented Data Abstraction

Die segmentorientierte Datenabstraktion macht sich typische Eigenschaften von Verkehrsinformationen zu Nutze. Es gibt immer eine *räumliche Komponente*, d. h. es wird eine bestimmte Situation an einem bestimmten Ort beschrieben. Die *Relevanz* der Verkehrsinformation ist an die räumliche Komponente gebunden, d. h. je weiter ein potentieller Empfänger einer Verkehrsinformationsnachricht von dem Ort des Geschehens entfernt ist, desto unwichtiger wird die Nachricht. Eine grundlegende Voraussetzung für die Abstraktion auf die o. g. Eigenschaften ist die Verwendung einer digitalen Karte. Die digitale Karte wird in einzelne Segmente einer bestimmten Länge[18] unterteilt und bietet in Kombination mit einer Straßen ID und der Fahrtrichtung des Fahrzeuges einen eindeutigen Schlüssel.

[18] Bspw. 100m lange Segmente für Landstraßen und 200m lange Segmente für Autobahnen

Jedes Fahrzeug kann nun neue Informationen für alle Segmente innerhalb der Sendereichweite aus eigenen und Fremd-Informationen generieren. In dem Prozess der Datenabstraktion wird dazu eine Funktion für alle K Informationselemente angewendet. Ein Informationselement ist von der Form $I_{s,k}$, mit $k \in [1,...,K]$, und wurde ausschließlich in einem Segment s empfangen. Das Verhalten der Funktion hängt von der Art der Anwendung ab. Beispielweise kann der Mittelwert aller $I_{s,k}$ berechnet oder das Maximum ausgewählt werden. Das Ergebnis wird dann in dem Fahrzeug persistent (zumindest bis zur nächsten Aktualisierung) als vorliegende Information zu Segment s gespeichert.

2. Schritt: Segment-Oriented Data Dissemination

Bei der segmentorientierten Datenverbreitung werden die Informationen, bezogen auf ein Segment, mittels Wireless Radio Transceiver übertragen. Fahrzeugantennen haben verglichen mit GSM Funkmasten eine eingeschränkte Sendereichweite. Da Informationen in einem SOTIS aber auch über weitere Strecken als die maximale Sendereichweite gesendet werden sollen, basiert die segmentorientierte Datenverbreitung auf den folgenden zwei Prinzipien:

Local Broadcast Die Anzahl der HOPs ist für alle Datenpakete auf 1 beschränkt. Dadurch können Knoten nicht direkt adressiert werden und es wird kein Routing - im klassischen Sinne über mehrere Knoten - von Paketen unterstützt. Gestärkt wird die Funktionsweise dieses Ansatzes durch die Tatsache, dass die von einem Fahrzeug in einem Segment gesammelten Daten meistens auch nur für Fahrzeuge in Sendereichweite interessant sind.

Application Layer Store-and-Forward Da alle Datenpakete wegen des Local Broadcast Prinzip nur einen HOP weit gesendet werden, ist die Anwendung für ein sinnvolles Weiterleiten von Informationen zuständig. Dabei sollten empfangene Informationen immer analysiert und mit den bereits vorliegenden (sofern vorhanden) eigenen Informationen verglichen werden. Nur wenn die neu empfangenen Informationen “besser” sind als die eigenen sollten diese beim periodischen Versenden der nächsten eigenen Pakete, mit als Information in die Auswahl einbezogen werden. Je wichtiger ein Segment markiert ist, desto häufiger wird es in Paketen berücksichtigt.

Zur Veranschaulichung der segmentorientierten Datenverbreitung wird Abbildung 2.8 näher betrachtet. t_0, t_1, t_2 und t_3 stellen den zeitlichen Verlauf von drei mit einem SOTIS ausgestatteten Fahrzeuge F_1, F_2 und F_3 zu auf einer in Segmenten unterteilten Straße dar. Fahrzeug F_1 und F_2 fahren in dieselbe Richtung, F_3 in die entgegengesetzte. Ferner ist $A(F_x,F_y)$ der Abstand zwischen den zwei Fahrzeugen F_x und F_y und D_{TX} die Funkreichweite aller Fahrzeuge. Als mögliches Szenario stellt Fahrzeug F_2 zum Zeitpunkt t_0 im aktuellen Segment eine Gefahr fest und warnt F_1 durch das Versenden der Information:

1. Da $A(F_1,F_2) >> D_{TX}$ ist, ist kein direktes Senden an F_1 möglich

2. F_3 liegt außerhalb D_{TX} von F_2
3. Da $A(F_2,F_3) < D_{TX}$ ist, empfängt und speichert F_3 die Nachricht von F_2
4. $A(F_1,F_3) >> D_{TX}$
5. $A(F_3,F_2) >> D_{TX}$
6. Da $A(F_1,F_3) < D_{TX}$ ist, empfängt F_1 die Nachricht von F_3 und ist somit vor der Gefahr gewarnt

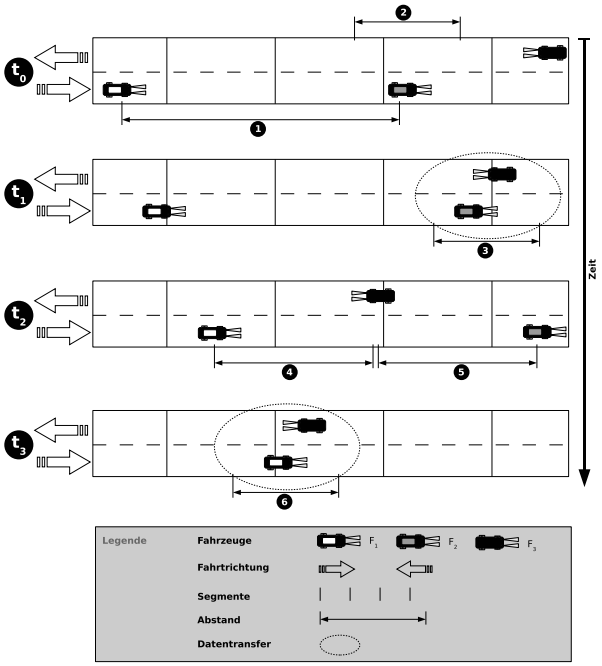

Abbildung 2.8: Datenverbreitung in einem SOTIS nach [WER04]

2.6.3 Architektur

Die Architektur von einem SOTIS Modell wird in [WER+03] mit insgesamt sieben Komponenten vorgestellt. Diese sind in Abbildung 2.9 skizziert und werden im Folgenden beschrieben:

SOTIS Core Der SOTIS Kern ist die zentrale Komponente des gesamten Systems. Er koordiniert die Berechnung der Verkehrsinformationen und entscheidet über die weitere Einbeziehung dieser Informationen in die von ihm generierten Datenpakete. Ferner überwacht er die Schnittstellen zu den *Sensoren*, der *Positionbestimmung*

und der *Kommunikationseinheit* um an Daten zu gelangen, die dann für weitere Berechnungen verwendet, gespeichert, versendet oder dem Benutzer auf dem Display angezeigt werden.

Knowledge Base Die Wissensbasis ist das Gehirn des SOTIS. Hier werden die für jedes Segment verfügbaren Informationen für ein Fahrzeug nach der Straßen-ID gespeichert. Zusätzlich wird ein Timestamp gespeichert der angibt, wann das Segment das letzte Mal versendet worden ist. In periodischen Abständen wird jedes Segment überprüft und bei einem geringen informativen Nutzen ggf. entfernt. Ein Ausschlusskriterium kann z. B. ein abgelaufener Zeitstempel sein.

Area Map Die Hauptfunktionen, die die Area Map bereitstellen sind die Zuordnung von Positionen zu Karten und die Segmentierung von Karten. Geographische Koordinaten können in ein Tripel aus $\langle \mathit{StraßenID}, \mathit{Segmentnummer}, \mathit{Richtung} \rangle$ konvertiert werden oder aus einem Tripel können wieder geographische Koordinaten entstehen.

Display Das Display ist die Schnittstelle zwischen Mensch und System. Je nach aktuell verfügbaren Informationen kann der momentane Standpunkt auf einer Karte oder alle sich in direkter Kommunikationsreichweite befindenden Fahrzeuge angezeigt werden.

Position & Sensor Die beiden Interfaces Position und Sensor haben sehr ähnliche Funktionen, weshalb sie oft auch als eine Komponente dargestellt werden. Als Schnittstelle zwischen einem SOTIS System und externen Geräten können sie die aktuelle Position dann von einer zuvor aufgezeichneten Tracedatei erhalten und an den Kern weitergeben. Weitere Möglichkeiten die aktuelle Position zu bestimmen ist die Verwendung einer Socketverbindung via TCP/IP oder das Abfragen von einem direkt angeschlossenen GPS Empfänger. Über den CAN Bus werden Fahrzeugdaten, wie z. B. die durchschnittliche Geschwindigkeit, Außentemperatur oder der aktuelle Status von dem Airbag und Elektronischem Stabilitätsprogramm (ESP), zur Verfügung gestellt.

Communication Eine grundlegende Eigenschaft eines dezentralen Verkehrssystems und somit auch eines SOTIS ist die Kommunikation mit der Außenwelt. Diese wird mit der Kommunikationsschnittstelle sichergestellt, die bspw. durch die Implementierung von IEEE 802.11p realisiert wird. Dieses Modul ermöglicht das Versenden eigener und Empfangen fremder Nachrichten.

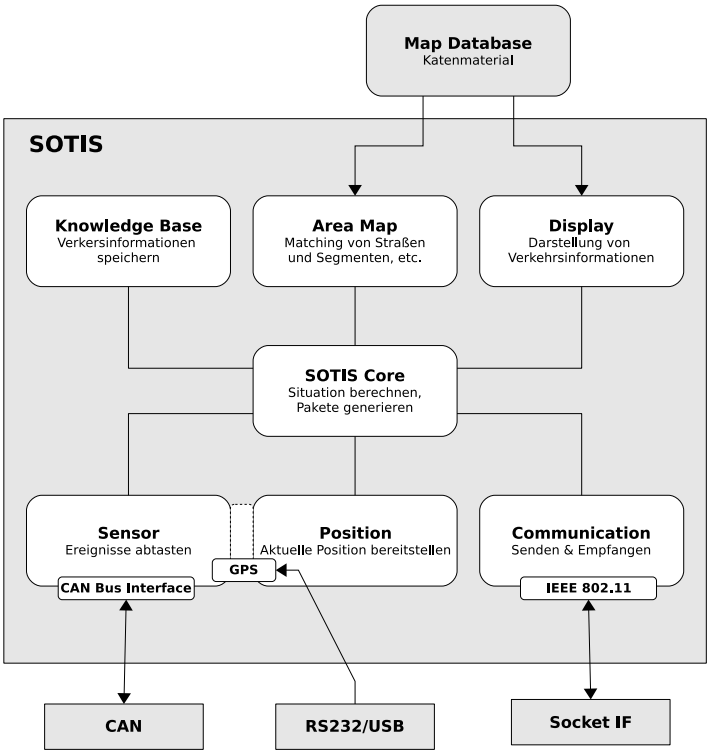

Abbildung 2.9: Architektur eines SOTIS Systems nach [WER04]

Kapitel 3

Architektur

Originalität wird überschätzt - Nachahmung ist die aufrichtigste Form, keine Dummheiten zu machen. [MPW07]

Nachdem am Ende des vorangegangenen Kapitels bereits kurz eine rudimentäre Architektur eines SOTIS Systems vorgestellt worden ist, beschäftigt sich dieses Kapitel nun - mehr globaler betrachtet - mit der Frage, was eine System-Architektur überhaupt ist und wie sie bei der Entwicklung von Systemen und Programmen verwendet werden kann und sollte.

Ganz allgemein betrachtet setzt sich der Begriff Architektur aus dem griechischen Wort *arché* für Anfang, Ursprung und dem Wort *techné* für Handwerk, Kunst, Fertigkeit zusammen (vgl. [HMGRZ06]). “Für manche von uns ist Architektur die Auswahl und der Einsatz einer Technologie, für andere ist Architektur vor allem ein Prozess, für viele ist Architektur eine Mappe mit Zeichnungen, auf denen miteinander verbundene geometrische Figuren zu sehen sind, für noch andere mag Architektur schlicht all das sein, was der Architekt produziert”. [VAC+05] Ein Blick in das Webster’s New World College Dictionary [Web09] liefert als klassische Definition von dem Begriff *architecture* die folgenden Beschreibungen[1]:

1. the science and art of designing and constructing buildings
2. a style of construction architecture
3. any framework, system
4. the design and interaction of components of a computer or computer system

Damit ist die Architektur einerseits Kunst (engl. art) andererseits auch eine Wissenschaft (engl. science), die sich mit dem Entwerfen (engl. designing) und Bauen (engl. constructing) von Gebäuden (engl. building) beschäftigt und dabei auch in unterschiedlichen Stilen (engl. style) Verwendung findet (vgl. [VAC+05]). Zusätzlich kann Architektur

[1] hier als gekürzte und zusammengefasste Version verwendet

ein beliebiges Framework oder System sein, das den Entwurf von interagierenden Komponenten repäsentiert. Eine eindeutige, kurze Definition ist somit kaum möglich, da der Begriff mit sehr vielen Bedeutungen überladen ist und je nach Kontext fachspezifisch ausgelegt wird.

Als wichtiges Charakteristikum einer Architektur im Sinne der Informatik nennt [VAC+05] eine überschaubare und handhabbare Komplexität. Es sollen nur wesentliche Aspekte eines Systems gezeigt werden, so dass es in kurzer Zeit möglich ist einen Gesamteindruck zu gewinnen, ohne sich in Details zu verlieren. Dennoch stellen Detailaspekte ein wichtiges Kriterium einer Architektur dar. Um einen Überblick von den "nicht trivialen Aufgaben einer Architektur" zu bekommen, wird in [VAC+05] eine Einordnung in verschiedene Dimensionen vorgenommen.

Im Folgenden werden die *Was*-, *Wo*-, *Warum*- und *Womit*-Dimensionen näher erläutert.

3.1 Architekturen und Architektur-Disziplinen (was)

Die *Was*-Dimension beschäftigt sich mit dem grundlegenden Architekturwissen und repräsentiert somit das Fundament für die unterschiedlichen Disziplinen von Architekturen. Neben den üblichen Anforderungen wie Verteilbarkeit, Verfügbarkeit, Wartbarkeit, Änderbarkeit, Wiederverwendbarkeit, Performance, Einfachheit, Kosten und hohe Integrierbarkeit soll eine Architektur der Herausforderung gewachsen sein, unterschiedliche Einflussfaktoren[2] für eine konkrete Problemstellung zu balancieren. Oft ist dafür aber Spezialwissen in bestimmten Bereichen notwendig, weshalb zwischen unterschiedlichen Architektur-Disziplinen differenziert werden kann (vgl. [VAC+05]):

Software-Architektur Eine Software-Architektur beschreibt die Software-Struktur bzw. Strukturen und die Software-Bausteine eines Systems. Des Weiteren werden die sichtbaren Eigenschaften[3] der Software-Bausteine sowie die Beziehungen untereinander festgelegt. Nach [PW92] ist es wichtig die tragenden Bausteine[4] einer Architektur zu identifizieren, da sie eine entscheidene Rolle für die erfolgreiche Umsetzung eines Systems spielen.

Enterprise-Architektur Eine Enterprise-Architektur legt den Fokus auf die Einhaltung und Umsetzung von Geschäftsstrategien, Informationen und Prozessen für IT-Architekturen von Unternehmen. Ferner kann sie dazu genutzt werden eine bestehende Ist-Architektur, anhand von festgelegten Standards und Richtlinien, in eine Ziel-Architektur zu überführen, indem dieser Übergangsprozess überwacht und gesteuert wird. Zu der Domäne einer Enterprise Architektur gehören u. a. Anwendungen, Daten und Technologien.

[2] Dazu gehören funktionale, qualitative und operative Aspekte

[3] Hierzu gehören angebotene Funktionalitäten, Schnittstellen oder auch Performance-Eigenschaften

[4] Dieses können Schnittstellen, Schlüsselklassen, Module, etc. sein. Siehe zusätzlich Abschnitt 3.4

Daten-Architektur Eine Daten-Architektur beschäftigt sich mit den Daten Aspekten eines Systems. Es werden logische und physische Datenmodelle festgelegt sowie Entscheidungen über die persistente Speicherung der Daten getroffen[5].

Integrations-Architektur Eine Integrations-Architektur hat die Aufgabe mehrere Anwendungen oder Systeme[6] zusammenzuführen. Da meistens heterogene Systeme, Plattformen oder Technologien vorliegen, ist eine ausgereifte Planung zur erfolgreichen Realisierung notwendig.

Netzwerk-Architektur Eine Netzwerk-Architektur befasst sich mit der Infrastruktur von Systemen. Dabei werden Funktionen, Dienste, Bausteine und Protokolle eines Netzwerkes geplant und konzipiert.

Sicherheits-Architektur Eine Sicherheits-Architektur verfolgt die Einhaltung von Vertraulichkeit, Integrität und Verfügbarkeit von Systemen. Neben der Authentifizierung und Authorisierung von Benutzern in Anwendungen, gehören auch die Planung und Umsetzung von Single Sign-on (SSO) und Public-Key-Infrastruktur (PKI) Implementierungen zu den Aufgaben dieser Architektur.

System-Management-Architektur Eine System-Management-Architektur beleuchtet operationale Aspekte von Systemen. Zu den wichtigsten Aufgaben gehören die Definition von Betriebsstrategien und Service Level Agreement (SLA) für zentrale und dezentrale Systeme bzw. Systemlandschaften.

In allen o. g. Architektur-Disziplinen spielt der Begriff des Systems eine zentrale Rolle. Dieses liegt nahe, da sich Architektur im allgemeinen Sinne mit Systemen aus unterschiedlichen Bereichen, wie Städte, Bauwerke, Landschaften oder IT-Systemen, beschäftigt. Nach [VAC+05] ist es dabei wichtig “die allgemeinen Eigenschaften von Systemen zu verstehen” und “ein grundlegendes Verständnis von Systemen und des Denkens in Systemen” zu haben. In der Systemtheorie wird zwischen *offenen* und *geschlossenen* Systemen unterschieden. Geschlossene Systeme kommunizieren nicht mit ihrer Umwelt, sind in der Praxis eher selten vertreten und werden aus diesen Gründen nicht weiter beschrieben. Offene Systeme hingegen kommunizieren und interagieren mit ihrer Umwelt. Weitere Begriffe aus der Systemtheorie sind *Emergenz*, *Holismus* und *Reduktionismus*:

Emergenz Nach [Rec91] ist das Ganze (System) mehr als die Summe seiner Einzelteile (Systembausteine). Demnach besitzt ein System Eigenschaften, die die einzelnen Systembausteine nicht besitzen, und erst durch das Zusammenführen und Interagieren der Systembausteine entstehen. Als Beispiel kann Wasser als Gesamtsystem betrachtet werden, welches bei Zimmertemperatur die Eigenschaft hat flüssig zu sein. Die einzelnen Wassermoleküle, als Systembausteine betrachtet, haben hingegen nicht die Eigenschaft flüssig zu sein.

[5] Bspw. Datenbank vs. Dateisystem
[6] Anwendungen / Systeme von einem oder mehreren Unternehmen

Holismus Der Holismus beschreibt den Ansatz ein System in einer ganzheitlichen Betrachtung zu sehen und zu verstehen. Dabei sollen die emergenten Eigenschaften des Systems erkannt werden, die durch das Zusammenwirken der Systembausteine entstehen (können). Abbildung 3.1 (links) zeigt diesen Ansatz, wobei die Subsysteme als Black Box betrachtet werden.

Reduktionismus Der Reduktionismus beschreibt den Ansatz ein System bzw. dessen Systembausteine getrennt voneinander zu betrachten. Abbildung 3.1 (rechts) repräsentiert den Reduktionismus, wobei das Subsystem B detailliert als White Box dargestellt ist.

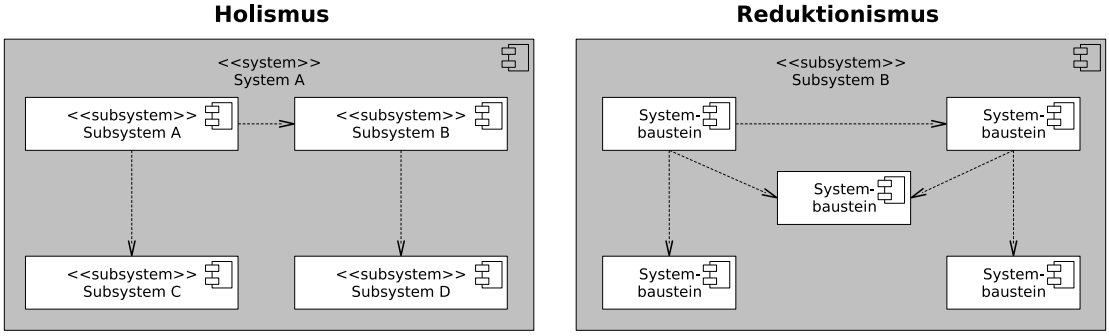

Abbildung 3.1: Komplementäre Ansätze der Systembetrachtung: Holismus und Reduktionismus nach [VAC+05]

Aus den oben beschriebenen Eigenschaften von Holismus und Reduktionismus wird klar, dass es sich um komplementäre Ansätze handelt. Nur wenn feststeht, aus welchen Systembausteinen ein System besteht (Reduktionismus), lassen sich auch Aussagen über das emergente Verhalten des Systems treffen (Holismus). Aus der Kombination von Systembausteinen mit Subsystemen bzw. Systemen lassen sich unterschiedliche Strukturen von Hierarchien beschreiben. Dabei können Subsysteme andere Subsysteme benötigen, Systembausteine aus anderen Systembausteinen bestehen und Systembausteine andere Systembausteine über Schnittstellen benutzen. [VAC+05]

3.2 Architektur-Perspektiven (wo)

Die *Wo*-Dimension beschäftigt sich mit den unterschiedlichen Ebenen und Sichten von Architekturen, um die Komplexität eines Systems in eine für den Menschen verarbeitbaren und überschaubaren Form zu repräsentieren. Denn psychologisch gesehen ist das menschliche Gehirn nur in der Lage 7 ± 2 Informationseinheiten (Items) gleichzeitig im Kurzzeitgedächtnis zu halten (vgl. [Mil56]).

3.2.1 Ebenen

Bei einem grundsätzlichen Architektur-Modell kann zwischen drei, miteinander verbundenen, Ebenen unterschieden werden (siehe Abbildung 3.2). Es gibt eine Organisationsebene, die mit einer Systemebene verbunden ist und die Systemebene ist wiederum mit der Bausteinebene verbunden. Dabei ist eine Architektur-Ebene jeweils von der direkt darüberliegenden abhängig. Die Organisationsebene beschäftigt sich hauptsächlich mit Geschäftsprozessen, IT-Standards und IT-Richtlinien auf höchstem Abstraktionsniveau von Organisationen und gehört somit zu der Architektur-Disziplin Enterprise-Architektur (siehe Abschnitt 3.1). Beispielsweise gibt sie als IT-Richtlinie die Verwendung von XML für den Datenaustausch in der gesamten Organisation vor, ohne dabei auf konkrete Technologien einzugehen. Dieses ist dann die Aufgabe der darunterliegenden Systemebene. Sie betrachtet alle Systeme der Organisation und beschreibt die verfügbaren Funktionalitäten. Um die Infrastruktur für Interoperabilität mittels Software-Framework für die verschiedenen Systeme einer Organisation zu gewährleisten, werden auf der Systemebene Architektur-Muster (siehe Abschnitt 3.4) verwendet. Auf dem niedrigsten Abstraktionsniveau befindet sich die Bausteinebene, die im Gegensatz zu den anderen beiden Ebenen nicht immer eindeutig einer Makro-Architektur[7] bzw. Mikro-Architektur[8] zugeordnet werden kann; obgleich sie zusammen mit der Systemebene zu der Architektur-Disziplin Software-Architekturen gehört (siehe Abschnitt 3.1). Da sich auf Bausteinebene auch Bausteine befinden können die Subsysteme enthalten, ist in Abbildung 3.2 ein “Ebenenwechsel” zu finden, der den rekursiven Charakter eines Subsystems als (weiteres Sub-) System (im Gesamt-System) skizziert. [VAC+05]

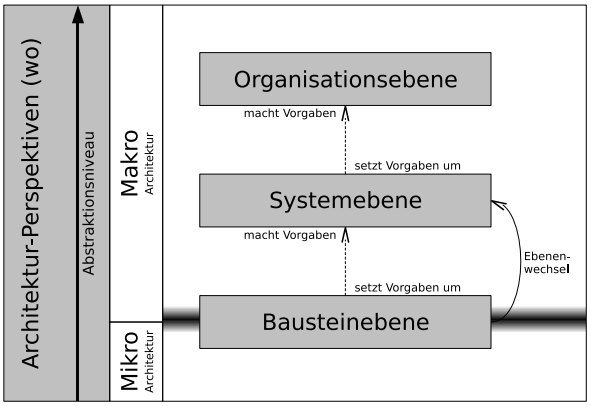

Abbildung 3.2: Modell der grundsätzlichen Architektur-Ebenen nach [VAC+05]

[7] ist Architektur im Großen und beschäftigt sich mit Entscheidungen und Strukturen einer Architektur auf hohem Abstraktionsniveau

[8] ist Architektur im Kleinen und beschäftigt sich mit dem Entwurf (nahe am Code)

3.2.2 Sichten

Wie auch die Ebenen haben die Sichten die Hauptaufgabe komplexe Systeme übersichtlicher darzustellen. Eine Sicht stellt ein vollständiges System aus dem Blickwinkel einer Menge von zusammenhängenden Interessen dar [IEE00]. Mit Interessen sind die unterschiedlichen Aspekte gemeint, die eine gute[9] Architektur abdecken muss. Als Beispiel für unterschiedliche Interessen kann einerseits die logische Sicht auf Bausteine, andererseits die physikalische Verteilung der Bausteine genannt werden. Betrachtet werden die Sichten mit Hilfe von Architektur-Modellen. Nach [VAC+05] sind Modelle eine Abstraktion des Originals, weil sie nicht sämtliche Details berücksichtigen. In Anlehnung an [BMMM95] und [BM04] werden die folgenden - Technologie unabhängigen - Architektur-Sichten genannt:

Konzeptionelle Sicht Die konzeptionelle Sicht beschreibt Bausteine und ihre Beziehungen. Auf die Darstellung von Details, wie z. B. Schnittstellen, wird bewusst verzichtet.

Logische Sicht Die logische Sicht spezifiziert Bausteine, ihre Beziehungen und Kommunikationsmechanismen im Detail, um diese später bei der technischen Realisierung zu verwenden.

Ausführungssicht Die Ausführungssicht beschreibt die physikalische Verteilung der Bausteine zur Laufzeit.

Die drei Sichten sind unvollständig und nicht ausreichend für eine “gute” Architektur, da beispielsweise Anforderungen und Daten nicht weiter betrachtet werden. Tabelle 3.1 zeigt ein erweitertes, abstraktes Architekturmodell welches kommerziellen Architektur-Sichten eine Grundlage bietet. Als kommerzielle Architektur-Sichten seien an dieser Stelle das *Zachman-Framework* [Zac87], das *Reference Model for Open Distributed Processing (RM-ODP)* [ISO98] (von der International Organization for Standardization (ISO) und Institute of Electrical and Electronics Engineers (IEEE) entwickelt) und das *4+1-Sichten-Modell* [MM01] (eine Ausprägung des RM-ODP) nur erwähnt und werden nicht weiter vorgestellt. [VAC+05]

3.3 Architektur-Anforderungen (warum)

Die *Warum*-Dimension beschäftigt sich mit Anforderungen, die den Ausgangspunkt einer Architektur bilden, und somit eine wichtige Rolle für die Entwicklung einer Architektur haben.

[9] Eine “gute” Architektur liegt vor, wenn die Komplexität reduziert worden ist, ohne dass architekturrelevante Aspekte fehlen (vgl. [VAC+05])

Architektur-Sicht	**Inhalt** (gekürzt) und anwendbare UML Diagramm-Typen
Geschäftssicht	• Wichtige, architekturrelevante (Geschäfts-) Prozesse und Anwendungsfälle • Fachliche Modelle • (Organisations-) Ziele und ihre Gegenstände • Nichtfunktionale Anforderungen UML AKD, AWD, KLD, PAD, SED, ZUD
Logische Sicht	• Modell der (Geschäfts-) Daten und ihrer Verarbeitung • Dekomposition eines Systems in Bausteine • Festlegung der Schnittstellen, Beziehungen und Verantwortlichkeiten von Bausteinen • Systemgrenzen • Szenarien wichtiger Anwendungsfälle UML AKD, KLD, KOD, KSD, PAD, SED, ZUD
Datensicht	• Tabellen und Felddefinitionen • Datenflüsse zwischen Bausteinen • Art der Daten, die zwischen Bausteinen ausgetauscht werden • Wichtige Datenquellen • Abbildung von Bausteinen auf persistente Daten UML KLD, KOD, PAD
Realisierungssicht	• Auswahl von Mitteln: Programmiersprachen, Software, Frameworks, Datenbanken, etc. • Festlegung und Richtlinien, wie die Mittel im Rahmen der geplanten Architektur verwendet werden sollen • Abbildung der Bausteine aus der logischen Sicht auf Code • Organisation und Verwaltung des Codes und sonstiger Ressourcen (Konfigurationsdaten, Grafiken, Web-Seiten, etc.) • Testen • Konfigurationsmanagement UML KLD, KOD, PAD, SED, VED, ZUD
Verteilungssicht	• Verteilung von Bausteinen auf Prozesse und physische Knoten • Installation und Konfiguration • Systembetrieb • Betriebsumgebung • Performance-Aspekte UML KOD, PAD, SED, VED, ZUD

Tabelle 3.1: Beschreibung eines abstrakten Architektur-Modells nach [VAC+05]

Nach [DT90] ist eine Anforderung “eine von dem Anwender benötigte Fähigkeit des Systems, um ein Problem zu lösen oder ein Ziel zu erreichen” und “eine Fähigkeit, die das System besitzen muss, damit es einen Vertrag, einen Standard, eine Spezifikation oder ein anderes formelles Dokument erfüllt”. Anders ausgedrückt sollen Anforderungen also genau das definieren, was eine Architektur und das damit (meistens) einhergehende System, leisten soll. Obwohl die Anforderungen zu Beginn der Architektur festgelegt werden, sollten diese immer möglichst präzise formuliert werden, damit sie die Eigenschaften *Korrektheit*, *Machbarkeit*, *Nachprüfbarkeit* und *Eindeutigkeit* erfüllen (vgl. [Wie03]). Die Korrektheit einer Anforderung kann nur von der Person überprüft werden, die die “wahren” Ziele der Anforderung kennt, wie bspw. der Auftraggeber. Die Machbarkeit einer Anforderung wird durch die Rahmenbedingungen oder auch technische bzw. physikalische Grenzen eingeschränkt. Konkrete Beispiele sind Budget und zeitliche Beschränkungen oder eine geforderte Bandbreite, die es aber nur theoretisch gibt. Eindeutigkeit bedeutet, die Anforderung so präzise wie möglich zu beschreiben, damit unterschiedliche Personen die dargestellte Information immer gleich interpretieren. Die Nachprüfbarkeit stellt sicher, dass die Anforderung nach Abschluss auch in dem System wiederzufinden ist und nachgewiesen werden kann. Deshalb ist es wichtig die Anforderung so zu formulieren, dass ein Nachweisen bzw. Messen möglich ist. Beispielsweise kann die Anforderung “die Abfrage von einem SQL Select Befehl auf genau eine Tabelle mit max. 5 Spalten und max. 60.000 Zeilen darf durchschnittlich nicht länger als 0,1 Sekunde dauern” gut überprüft werden. [VAC+05]

Ein System wird durch eine Menge von Anforderungen - ergo einem Anforderungskatalog - beschrieben. Dieser Anforderungskatalog sollte zum einen *vollständig*, zum anderen *konsistent* sein (vgl. [VAC+05]). Vollständig bedeutet in diesem Zusammenhang ein möglichst abgerundetes Bild von dem Gesamtsystem durch den Anforderungskatalog erfasst zu haben. Dieses beinhaltet sowohl die Standardabläufe als auch zu erwartende Fehlersituationen wie z. B. falsche oder nicht vorhandene Informationen. Die Konsistenz soll widersprüchliche Anforderungen vermeiden, indem diese kritisch hinterfragt werden.

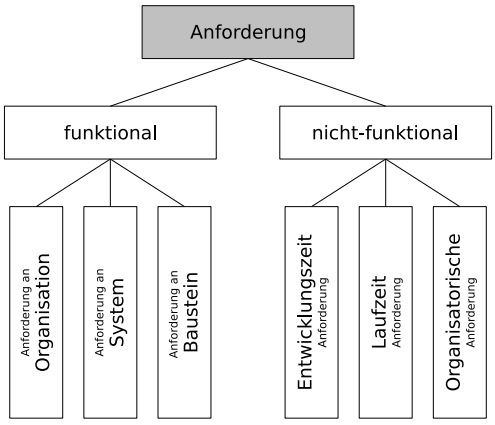

Abbildung 3.3: Klassifikation von Anforderungen

Abbildung 3.3 zeigt die Möglichkeit Anforderungen zu klassifizieren. Grundlegend kann zwischen *funktionalen* und *nicht-funktionalen* Anforderungen unterschieden werden. Funktionale Anforderungen legen fest, was das System tun soll, die nicht-funktionalen Anforderungen legen fest, welche Eigenschaften das System haben soll (vgl. [RR06]).

Bezogen auf das in Abschnitt 3.2 vorgestellte Architektur-Ebenen Modell können funktionale Anforderungen weiter in Organisations-, System- und Bausteinanforderungen unterteilt werden. Funktionale Organisations-Anforderungen sind funktionale Anforderungen die an eine Organisation selbst gestellt werden. Als Beispiel können hier Mitarbeiter oder auch Kunden genannt werden, die bspw. die Anforderung haben “Bestellungen bei der Organisation durchzuführen”. Diese Meta-Meta Anforderung ist Grundlage für funktionale System-Anforderungen, die die Wünsche konkretisiert und durch die Meta-Anforderung “eine Bestellung soll in einem Auftragssystem erfasst werden” näher spezifiziert. Basierend auf dieser Meta-Anforderung können nun funktionale Baustein-Anforderungen festgelegt werden. Im Fall unseres Beispiels sollte dann festgehalten werden, dass der Baustein “auf einen weiteren zugreift, um einen Auftrag in die Datenbank zu schreiben”. Auch nicht-funktionale Anforderungen können weiter unterteilt werden in Entwicklungszeit-, Laufzeit- und organisatorische Anforderungen. Entwicklungzeit-Anforderungen beschreiben Eigenschaften die während der Entwicklung eines Systems zu berücksichtigen sind und somit indirekt die Qualität mit beeinflussen. Zu den klassischen Eigenschaften zählen u. a. Erweiterbarkeit, Wiederverwendbarkeit und Plattformunabhängigkeit sowie Aspekte einer zugrunde liegenden Ziel-Technologie. Laufzeit-Anforderungen beziehen sich auf die Eigenschaften des Systems während des Betriebs. Häufig sind hier Verfügbarkeit, Stabilität und Performance gängige Anforderungen. Als letzte Klasse gibt es die organisatorischen Anforderungen. Diese legen oft das Budget oder einen Fertigstellungstermin fest oder tragen Sorge dafür, dass die von der Organisation festgelegten IT-Richtlinien bei der Umsetzung eingehalten werden. [VAC+05]

3.4 Architektur-Mittel (womit)

Die *Womit*-Dimension beschäftigt sich mit Prinzipien, Mustern und Strukturen von Architekturen und stellt somit zentrale Konzepte und Techniken zur Erarbeitung, Beschreibung und Umsetzung einer Architektur bereit.

3.4.1 Prinzipien

Bei der Erstellung einer Architektur sollen Prinzipien dabei helfen diese ”gut zu gestalten“. Wie wir bereits schon in den vorangegangenen Abschnitten erfahren haben, ist es schwer zu beurteilen, was eine ”gute“ Architektur ausmacht. Folgend werden Prinzipien vorgestellt, die sich bei der Verwendung von Architekturen in der Praxis bewährt haben. [VAC+05]

Prinzip der losen Kopplung Ein Software-System besteht i. d. R. aus mehreren interagierenden Bausteinen (siehe Abschnitt 3.1). Die Bausteine bestehen wiederum aus verfeinerten granularen Konstrukten wie z. B. Klassen. Wenn die Klassen in Beziehung zueinander stehen, sind sie von einander abhängig und somit gekoppelt. Bei der Betrachtung von zwei gekoppelten Klassen kann bspw. zwischen unterschiedlichen Stärken der Kopplung unterschieden werden: Klassen sind *stark* gekoppelt, wenn sie gegenseitig auf ihre privaten Daten zurückgreifen. Sie sind *weniger stark* gekoppelt, wenn sie über eine gemeinsame Datenstruktur kommunizieren und *gering* gekoppelt, wenn sie über Methodenparameter kommunizieren. Das Prinzip der losen Kopplung besagt, dass die Kopplung zwischen System-Bausteinen möglichst gering gehalten werden soll, damit bei Änderungen nicht zu viele Abhängigkeiten mit berücksichtig werden müssen (vgl. [YC87]). Erreicht werden kann dieses Ziel durch weitere Prinzipien *Abstraktion*, *Separation of Concerns* und *Information Hiding*. Somit sollten alle Informationen grundsätzlich nur über gemeinsame Schnittstellen ausgetauscht werden, die zudem möglichst wenige Elemente enthalten. Zusätzlich sollten zirkuläre Abhängigkeiten[10] von Bausteinen vermieden werden. In der Regel wird diese durch das Hollywood-Prinzip *"Don't call me, I call you."* [WT05] erreicht.

Prinzip der hohen Kohäsion Da System-Bausteine meist selbst aus vielen Teilen bestehen, wie z. B. Klassen, Variablen und Methoden, liegen auch dort Abhängigkeiten innerhalb des Bausteins vor. Im Gegensatz zur losen Kopplung ist hier eine hohe Abhängigkeit besonders gut, da sich relevante Eigenschaften in einer Beschreibung bündeln lassen. Es liegt eine hohe Kohäsion vor, wenn sich ein Objekt x (als Instanz der Klasse x, mit Variablen und Methoden) zur Laufzeit möglichst oft selbst aufruft (vgl. [YC87]). Global betrachtet stehen die lose Kopplung und hohe Kohäsion in einer gegenseitigen Wechselwirkung (siehe Abbildung 3.4).

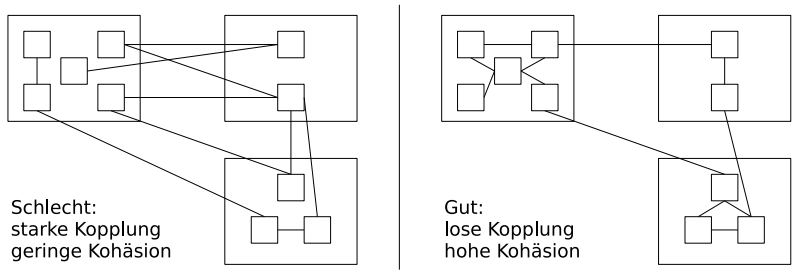

Abbildung 3.4: Wechselwirkung von Kopplung und Kohäsion nach [VAC+05]

Separation-of-Concerns Prinzip Das Separation-of-Concerns Prinzip (zu deutsch: Trennung von Aufgabenbereichen/Belangen) ist, in Bereichen außerhalb der

[10] Minimalbeispiel: Baustein A holt sich Informationen von Baustein B und Baustein B sendet Informationen an Baustein A

Software-Architektur auch unter der Bezeichnung Divide and Conquer zu finden, und beschreibt den in Abschnitt 3.1 vorgestellten Reduktionismus als Lösungsansatz. Die Herausforderung bei der Verwendung von diesem Prinzip ist es, geeignete Aspekte bzw. Aufgaben des Systems zu finden, anhand derer eine Teilung des Gesamt-Systems in einzelne Teile vorgenommen werden kann. Ein häufig verwendetes Kriterium ist die Funktionalitätsanforderung, die bspw. die Bausteine eines Systems nach ihren spezifischen Funktionalitäten gliedert (Dekomposition[11]). Weitere Kriterien können Wiederverwendbarkeit, Performance oder Ressourcenverbrauch sein.

Information-Hiding Prinzip Bei dem Information-Hiding Prinzip (zu deutsch: verbergen von Informationen) werden Entwurfsentscheidungen in Systembausteinen verborgen, die nach außen durch festgelegte Schnittstellen angesprochen werden sollen. Dieses Prinzip wird nach David Parnas (vgl. [Par71, Par72]) auch Datenkapselung genannt und wird im objekt-orientierten Kontext mit `public`, `private`, `package` und `protected` Sprachkonstrukten realisiert. Konkret wird dieses Prinzip als Facade-Entwurfsmuster (vgl. [GHJV95]) zur Strukturierung von Clients mit Zugriff auf ein Subsystem beispielhaft in Abbildung 3.5 dargestellt. Die jeweiligen Schnittstellen werden nur noch an einer Stelle - der Facade - definiert, und sind somit Änderungen gegenüber lokal wartbar. Weitere Anwendung findet das Information-Hiding Prinzip bei einem Schichtenmodell, bei dem Schicht n jeweils auf die darunter liegende $n-1$ Schicht über Schnittstellen zugreift.

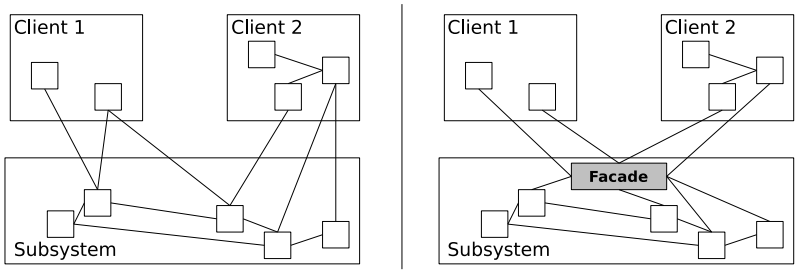

Abbildung 3.5: Information Hiding Prinzip ohne und mit Facade nach [GHJV95]

Abstraktionsprinzip Das Prinzip der Abstraktion ist wichtig, um eine gegebene Komplexität von einem Problem weiter herunterzubrechen. ”Eine Abstraktion gibt die wesentlichen Charakteristika eines Objektes an, die es von allen anderen Arten von Objekten unterscheiden, wobei klar definiert konzeptuelle Grenzen gesetzt werden, und zwar unter Bezugnahme auf die Perspektive des Betrachters“ [Gra97]. Dies

[11] Bsp.: Bei einem System, mit dem Benutzer über eine Benutzerschnittstelle mit einer Artikel-Datenbank interagieren, werden die Bausteine Benutzerschnittstelle, Bestellabwicklung, Datenbankzugriff allgemein, Datenbankzugriff Artikeldatenbank, Datenbankzugriff Lieferantendatenbank und Datenbankzugriff Bestelldatenbank verwendet

bedeutet, dass ein Objekt von unterschiedlichen Betrachtern unterschiedlich abstrahiert wird.

Modularitätsprinzip Das Modularitätsprinzip besagt, dass System-Bausteine leicht austauschbar und in sich abgeschlossen sein sollen. Die Einhaltung wird durch die Verwendung der Prinzipien Separation-of-Concerns und Information-Hiding begünstigt und geht mit der Umsetzung der Prinzipien der losen Kopplung und hohen Kohäsion einher.

3.4.2 Muster

”Ein Muster ist eine dreiteilige Regel, die die Beziehungen zwischen einem bestimmten Kontext, einem Problem und einer Lösung ausdrückt“ [AIS+77]. Diese aus dem Jahre 1977 stammende Definition ist recht allgemein gehalten und soll dabei den Grundgedanken widerspiegeln, dass Probleme nach ihrem Kontext kategorisiert werden, auf welche dann etabliertes Wissen (das Muster) als Lösung angewendet werden kann. [BMR+98] definiert in diesem Zusammenhang ”ein Softwarearchitektur-Muster als ein häufig, in einem speziellen Entwurfskontext auftretendes Entwurfsproblem, das ein erprobtes generisches Schema zur Lösung präsentiert“. Der Kontext beschreibt dabei die Situation, in der das Problem auftritt und ist durch die große Anzahl von unterschiedlichen Situationen nicht einfach zu spezifizieren. Das Problem selbst wird nach [AIS+77] durch sog. *Kräfte* (engl. *forces*) beschrieben und bspw. in Anforderungen oder Rahmenbedingungen festgehalten. Die Lösung zeigt dann, wie das aufgetretene Problem durch ausbalancieren der Kräfte gelöst werden sollte, in Form von Konfigurationen, welche die Strukturen von Komponenten bzgl. Beziehungen und auf Laufzeitverhalten beschreiben. In Abbildung 3.6 läßt sich erkennen, dass Kräfte in eine Richtung wirken können (Erweiterbarkeit und Wartbarkeit) oder auch in entgegengesetzte Richtungen (Erweiterbarkeit und Minimale Code-Größe).

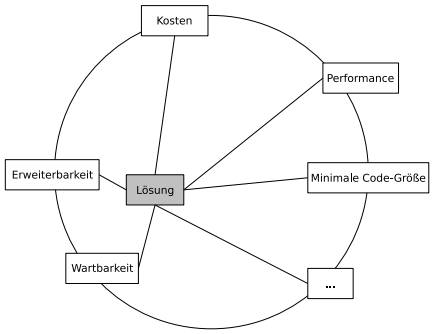

Abbildung 3.6: Balancieren von Kräften

Muster beziehen sich auf unterschiedliche Ebenen eines Systems und werden nach [BMR+98] analog zu Abschnitt 3.2 in die folgenden drei Gruppen eingeteilt[12]:

[12] Hinweis: In der Literatur gibt es noch viele andere Ansätze, Muster zu kategorisieren

Architekturmuster Architekturmuster sind Muster auf höchster Ebene, die die fundamentale Struktur von einem Softwaresystem beschreiben. Dabei wird jede Komponente bzw. Subsystem des Softwaresystems durch einen Zuständigkeitsbereich und Regeln mit Beziehungen zu anderen Subsystemen fokussiert. "Architekturmuster sind Schablonen (engl. *templates*) für konkrete Software-Architekturen" [BMR+98], sollen als Grundgerüst des Softwaresystems fungieren und zu Beginn des Grobentwurfs verwendet werden.

Entwurfsmuster Eine Abstraktionsebene unter den Architekturmustern befinden sich die Entwurfsmuster. "Ein Entwurfsmuster beschreibt ein Schema zur Verfeinerung von Subsystemen oder Komponenten eines Softwaresystems oder den Beziehungen zwischen ihnen" und "hat keine Auswirkung auf die grundsätzliche Struktur, kann aber einen großen Einfluß auf die Architektur eines Subsystems haben" [BMR+98]. Sie sollen dem in Abschnitt 3.4.1 vorgestellten Prinzip der losen Kopplung gerecht werden, sind meistens Programmiersprachen unabhängig und werden am Ende des Grobentwurfs verwendet.

Idiome Idiome befinden sich auf der untersten Abstraktionsebene und beschreiben mögliche Implementierungen von Entwurfsaspekten in einer konkreten Programmiersprache. Eingesetzt werden Idiome meist erst in der Implementierungsphase. Somit kann ein Idiom für eine C++ Implementierung, die Objektreferenzen verwendet, grundlegend von einem Idiom für eine Smalltalk Implementierung verschieden sein, da in Smalltalk bspw. der Speicher automatisch bereinigt wird (vgl. [BMR+98]).

Muster stellen also in unterschiedlichen Phasen und auf unterschiedlichen Ebenen Lösungsansätze bereit, die auf Erfahrungen und Vorgehensweisen von anderen, ähnlichen Problemen basieren. Um ein gegebenes Problem innerhalb eines Kontextes zu identifizieren werden in [BMR+98] Kriterien zur Klassifikation von Mustern vorgestellt. Tabelle 3.2 zeigt die Kriterien der unterschiedlichen Arten von Mustern und ordnet sie konkret existierenden Mustern zu. Dabei ist eine eindeutige Abgrenzung nicht immer gegeben, was sich bspw. durch das doppelte Vorkommen des *Pipes-and-Filters* Musters zeigt. Es gibt eine Vielzahl von weiteren Mustern für viele unterschiedliche Bereiche. Ein guter Überblick ist unter [Arc09], [Wik09] und [HMGRZ06] zu finden.

3.4.3 Dokumentationsmittel

Dokumentationsmittel haben das Ziel architektonische Entscheidungen und Strukturen, die anhand der bisher vorgestellten Mittel, wie bspw. den Prinzipien und Mustern auf den unterschiedlichen Ebenen und Sichten erarbeitet wurden, festzuhalten. Das dokumentierte Wissen stellt für den gesamten Zeitraum der Realisierung, von dem Grobentwurf bis hin zur Fertigstellung eines Software Systems, ein unverzichtbares Instrument dar. Die Unified Modeling Language (UML) ist nach [VAC+05] eines der wichtigsten Dokumentationsmittel. Durch die Vielzahl an unterschiedlichen Diagrammtypen bietet sie ein breit gefächertes Repertoire an. So läßt sich UML dazu verwenden alle in Tabelle 3.1 befindlichen

	Kriterium	Muster
Architekturmuster	Vom Chaos zur Struktur *Zerlegung einer Gesamtaufgabe in Teilaufgaben*	Layers Pipes-and-Filters Blackboard
	Verteilte Systeme *Infrastruktur für Systeme, deren Komponenten in verschiedenen Prozessen oder Adressräumen liegen*	Broker Pipes-and-Filters Microkernel
	Interaktive Systeme *Strukturierung von Systemen mit Mensch-Maschine-Interaktion*	Model-View-Controller Presentation-Abstraction-Control
	Adaptive Systeme *Infrastruktur für erweiterbare und sich weiterentwickelnde Systeme*	Microkernel Reflection
Entwurfsmuster	Strukturelle Zerlegung *Geeignetes Zerlegen von Subsystemen und komplexen Komponenten in miteinander kooperierende Einheiten*	Whole-Part
	Organisation von Arbeit *Struktur für die Zusammenarbeit mehrerer Komponenten, zur Bereitstellung einer komplexen Dienstleistung*	Master-Slave
	Zugriffskontrolle *Zugriff auf Komponenten und Dienste schützen und kontrollieren*	Proxy
	Management *Homogene Mengen von Objekten, Diensten und Komponenten in ihrer Gesamtheit behandeln*	Command-Processor View-Handler
	Kommunikation *Organisation von Kommunikation zwischen Komponenten*	Publisher-Subscriber Forwarder-Receiver Client-Dispatcher-Server
Idiome	Ressourcen Verwaltung *Verwaltung gemeinsam genutzter Komponenten und Objekte*	Counted-Pointer

Tabelle 3.2: Klassifizierung von Mustern nach [BMR+98]

Sichten in unterschiedlichen Diagrammtypen zu repräsentieren und somit unterschiedlichen Abstraktionsebenen gerecht werden. Da selbst eine gekürzte Vorstellung der UML den Rahmen dieses Kapitels bei weitem sprengen würde, wird das Wissen über gängige Diagrammtypen von nun an implizit als gegeben angenommen.

3.4.4 Architektur-Strukturen

Wie bei den Mustern gibt es auch eine Reihe von Strukturen, die sich bei Architekturen bewährt haben. Zu den grundlegenden Arten gehören u. a. n-Tier-Architekturen, Rich Client[13] & Thin Client, Kommunikations-Middleware-Architekturen, Serviceorientierte Architekturen (SOA) und Peer-to-Peer-Systeme (P2P-Systeme).

”n-Tier-Architekturen beschreiben Modelle zur Verteilung einer Anwendung auf den Knoten eines verteilten Systems“ [Ham05]. Unterschiedliche Aufgabenbereiche und Zuständigkeiten werden so *n* unterschiedlichen Schichten[14] zugeordnet. Es gibt 2-Tier-Architekturen, die aus einem Client und einem Server bestehen und durch ein Anfrage-/Antwort-Schema kommunizieren, 3-Tier- und mehr-Tier-Architekturen. Eine 3-Tier-Architektur ist in Abbildung 3.7 zu sehen und zeigt die Verteilung der drei Aufgaben *Präsentation*, *Anwendungslogik* und *Datenhaltung*. Die Client-Schicht (Tier 1) steuert

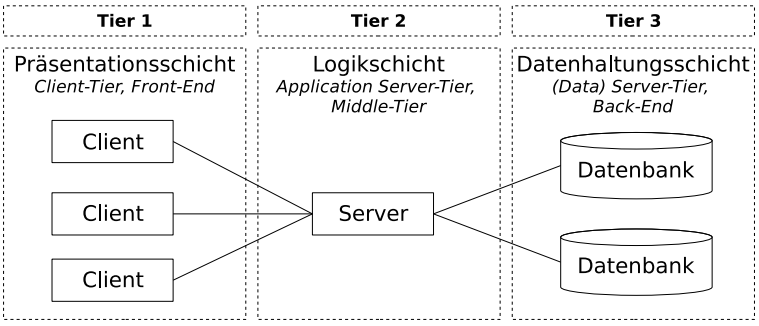

Abbildung 3.7: Aufbau einer 3-Tier-Architektur

die Interaktion mit dem Benutzer, repräsentiert Daten und nimmt Events entgegen. Die Middle-Schicht (Tier 2) stellt der Client-Schicht die Logik der Anwendung zur Verfügung. Es gibt Standardlösungen, wie Transaktionsmonitore, Messaging Server und Application Server durch deren Einsatz bei einer hohen Anzahl von Clients eine Steigerung der Flexibilität erreicht wird (vgl. [VAC+05]). Bei der Datenhaltungsschicht (Tier 3) werden meistens Datenbanksysteme zum persistenten Speichern und Bereitstellen der Daten verwendet. Durch die Aufteilung in die drei Schichten lassen sich einige in Abschnitt 3.4.1 vorgestellten Prinzipien in der 3-Tier-Architektur wiederfinden.

[13] In der Literatur auch unter der Bezeichnung *Fat Client* zu finden
[14] engl. *tier*

	Rich-Client	Thin-Client	2-Tier	3-Tier
Netzwerkverbindung				
gut		●	●	●
schlecht	●		●	
Komplexität Anwendungslogik				
hoch		●		●
gering	●		●	
Nebenläufigkeit				
hoch		●		●
gering	●	●	●	

● = trifft zu

Tabelle 3.3: Richtlinien zum Einsatz von Architektur-Typen nach [Ham05]

In direktem Zusammenhang mit einer n-Tier-Architektur steht der Rich-Client bzw. Thin-Client und beschreibt, wie viel Funktionalität ein Client enthalten soll. Ein Thin-Client enthält wenig Funktionalität, und stellt lediglich die vom Server berechneten Ergebnisse dar. Der Rich-Client hingegen entlastet den Server und übernimmt zahlreiche Funktionalitäten selbst. Die Vor- und Nachteile von den Client-Typen bezogen auf 2- und 3-Tier-Architekturen sind in Tabelle 3.3 zusammengestellt.

Kommunikations-Middleware-Architekturen beschäftigen sich mit Verteilungsarchitekturen einer Client-/Server Architektur und widmen sich den damit einhergehenden Problemen der Vorhersagbarkeit und Latenz eines Netzwerkes, Nebenläufigkeit von Prozessen, und Skalierbarkeit des gesamten Systems. Die Kommunikations-Middleware ist eine zusätzliche Softwareschicht, die sich oberhalb der Netzwerk API und unterhalb der Anwendungsschicht befindet und einem Entwickler die Kommunikationsaufgaben abnimmt (vgl. [VAC+05]). Technisch umgesetzt werden kann der Kommunikationsmechanismus einer Kommunikations-Middleware-Schicht durch die Verwendung von Remote Procedure Call (RPC) (vgl. [VKZ04]).

Eine weitere Struktur, die seit kurzer Zeit immer mehr an Bedeutung gewonnen hat, sind SOA. Oft werden SOA mit Web-Services und den Begriffen *Simple Object Access Protocol (SOAP)*, *Universal Description, Discovery and Integration (UDDI)* und *Web Service Description Language (WSDL)* auf die gleiche Ebene gestellt, was nicht korrekt ist. SOA ”sind eine spezielle Architektur-Struktur, die beschreibt, wie man Middleware nutzen kann, um lose gekoppelte Services zu erreichen“ [VAC+05] und stellen ein Konzept dar. Der Web-Service an sich, kann als konkrete Umsetzung von SOA betrachtet werden und verwendet dabei WSDL zum Austausch XML-basierter Nachrichten zwischen einzelnen Web-Services, SOAP als Kommunikationsprotokoll und UDDI als Verwaltungsdienst für die Web-Services. Im Mittelpunkt stehen also die Services[15], die als eine Men-

[15] Die Begriffe *Service(s)* und *Dienst(e)* werden hier synonym verwendet, sind aber von dem Begriff *Web-Service(s)* abzugrenzen

ge von Bausteinen gesehen werden können und auf einer niedrigen Abstraktions-Ebene Funktionalitäten kapseln und miteinander agieren. Auf einer höheren Abstraktions-Ebene werden die Bausteine dann orchestriert[16] und zu einem Service zusammengefasst. Ferner werden so ”vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht“ [M$^+$07]. Das technische Zusammenspiel von den unterschiedlichen Rollen *Service-Anbieter*, *-Verzeichnis* und *-Nutzer* innerhalb SOA ist in Abbildung 3.8 skizziert. SOA

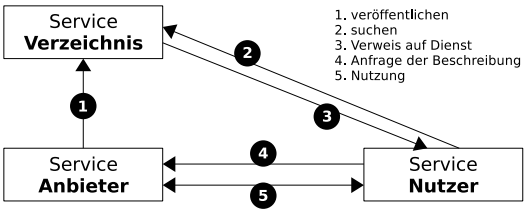

Abbildung 3.8: SOA Rollen und Aktionen nach [M$^+$07]

folgen den in Abschnitt 3.4.1 vorgestellten Prinzipien und ermöglichen so eine Reihe von Vorteilen, die in Tabelle 3.4 zusammengefasst sind.

Im Gegensatz zu allen bisher vorgestellten Strukturen stellen die P2P-Systeme einen komplimentären Ansatz dar, da ihnen weder ein Client-/Server noch eine n-Tier-Architektur zu Grunde liegt. Bei Peer-to-Peer-Systemen (P2P-Systemen) sind alle Teilnehmer gleichberechtigt - es gibt keinen Server, nur Clients. Analog zu dem Service-Verzeichnis bei SOA müssen die Clients allerdings einen Lookup-Service anbieten und fungieren somit indirekt wieder als Server.

3.5 Zusammenfassung

In diesem Kapitel wurden u. a. eine Reihe von Prinzipien, Sichtweisen, Strukturierungsmerkmalen und Strukturen von Architekturen kurz vorgestellt. Da eine Architektur immer nur einen bestimmten Ausschnitt von einem abstrahierten Bereich der Realität repräsentiert, ist es wichtig zu wissen ***was*** *wo*, *warum* und *womit* zu beachten ist. Abschließend und vorausschauend für die folgenden Kapitel soll festgehalten werden, was eine gute Architektur ausmacht: Schnittstellen sollen klein gehalten werden, damit diese einfach zu verwenden und zu pflegen sind. Code soll nach Änderungen der Anforderungen an möglichst wenigen Stellen angepasst werden. Klassen sollen nur gleiche Anforderungen abdecken, damit sie übersichtlich bleiben.

[16] Orchestrierung beschreibt die ausführbaren Aspekte eines (Geschäfts-) Prozesses (vgl. [M$^+$07])

SOA-Prinzip	Beschreibung	Vorteile
Dekomposition und Kapselung	Geschäftsprozesse in kleine Einheiten zerlegen; Kapseln von Funktionalitäten; Komponentenbildung und Servicedefinition	• Flexibilität durch kleinere (Fach-) Komponenten
Abstraktion	Servicedefinition erfolgt in Servicebeschreibung als Abstraktionsschicht für Servicenutzer; Implementierung des Services für Nutzer nicht sichtbar und irrelevant	• Komplexitätsreduzierung • Flexibilität
Offenheit und Standardisierung	Allgemeine, offene Standards verwenden; Bsp.: WSDL, SOAP	• Austauschbarkeit • Transparenz
Lose Kopplung	Autonomie der verbundenen Services und Verminderung von Abhängigkeiten; flexibel kombinierbar	• Reduktion der Abhängigkeiten • Komplexitätskontrolle • Austauschbarkeit
Komposition	Verkettung und Orchestrierung von wiederverwendbaren Services zu (Teil-) Prozessen; austauschbar und Veränderung der Reihenfolge möglich	• Flexibilität • Austauschbarkeit • Anpassungsfähigkeit bzgl. neuen Anforderungen

Tabelle 3.4: SOA-Prinzipien und Vorteile nach [HL07]

Kapitel 4

Analyse und Auswahl einer Architektur

Die Entwicklung einer Architektur ist ein sehr komplexer Prozess, bei dem viele Tätigkeiten miteinander verwoben sind. [RH09]

Das bisher in den vorangegangenen Kapiteln erarbeitete Wissen über Architektur und Verkehrsinformationssysteme wird nun verwendet, um die Grundsteine für das Fundament einer flexiblen SOTIS-Architektur zu legen. Dazu werden in einem ersten Schritt unterschiedliche Szenarien vorgestellt, in denen ein SOTIS verwendet wird, um daraus in einem zweiten Schritt wichtige Anforderungen zu abstrahieren. “Der primäre Nutzen bei der Festlegung solcher Abstraktionen ist, dass sie unser Problem eingrenzen; sie betonen die Dinge, die sich im System befinden und deshalb relevant für das Design sind [..] .” [Gra97] In einem weiteren Schritt werden die Anforderungen bewertet, gängigen mobilen Plattformen gegenübergestellt und abschließend zur Auswahl einer Plattform verwendet.

4.1 Szenarien - Anwendungsgebiete eines SOTIS

Die in diesem Abschnitt beschriebenen Szenarien haben das Ziel die Funktionen bzw. Aufgaben eines SOTIS in unterschiedlichen Anwendungsgebieten darzulegen, um so Rückschlüsse auf eine Architektur zu ziehen. [Gra97] empfiehlt “die Verwendung von Szenarien, um den Prozeß voranzutreiben”. Wegen der großen Anzahl von unterschiedlichen Situationen, die im Straßenverkehr entstehen können, wurden die Szenarien möglichst heterogen gewählt und erheben - realitätsbedingt - keinen Anspruch auf Vollständigkeit.

4.1.1 Szenario I - Auf der Autobahn

Alice ist mit ihrem Personenkraftfahrzeug zügig auf einer freien Autobahn unterwegs, der Route ihres Navigationssystems folgend. Hinter einer, für Alice nicht einsehbaren Kurve, die sie bald passieren wird, haben Bob und Eve stark abbremsen und ausweichen müssen,

da sich ein Vierkantholz auf der Fahrbahn befindet. Das Abbremsen und Ausweichen werden von Bobs und Eves SOTIS erkannt und als Information an Alice gesendet. Da Alice Fahrzeug auch mit einem SOTIS ausgestattet ist, werden die Informationen von Bob und Eve ausgewertet und Alice wird über ihr Display informiert, dass sich vor ihr ein Hindernis befindet. Vor Erreichen der Gefahrenstelle reduziert Alice ihre Geschwindigkeit und gelangt sicher an dem Hinderniss vorbei.

4.1.2 Szenario II - In der Stadt

Bob befindet sich in SOTIS-City[1] und steht vor einer roten Ampel. Während er wartet, erhält sein SOTIS Verkehrsinformationen von allen anderen, sich in der Nähe befindenden Teilnehmern. Die Anzahl an Verkehrsinformationen ist dementsprechend hoch. Die Ampel wird grün und Bob fährt weiter. Nach 200m wird er informiert, dass direkt vor ihm ein Stau liegt.

4.1.3 Szenario III - Unfall

Es ist Winter. Trudy fährt mit ihrem Kraftfahrzeug auf einer Landstraße. Sie fährt mit überhöhter Geschwindigkeit in eine Kurve und verliert aufgrund von Glatteis die Kontrolle über ihr Fahrzeug, welches sich mehrfach überschlägt und auf dem Dach liegen bleibt. Der Neigungssensor des Fahrzeuges stellt den aktuellen Wert dem SOTIS bereit. Das SOTIS entscheidet, dass ein Unfall vorliegt und sendet die Position und die Information “Unfall mit Überschlag” aus. Andere Verkehrsteilnehmer mit SOTIS empfangen die Nachricht und sind einerseits vor der Gefahrenstelle gewarnt und können andererseits erste Hilfe leisten.

4.2 Anforderungen durch den Einsatzzweck

Aus den beschriebenen Szenarien werden folgend Anforderungen für eine Architektur bzw. die Funktionen eines SOTIS abstrahiert. Dabei handelt es sich, um die in Abschnitt 3.3 vorgestellte Klasse der nicht-funktionalen Entwicklungszeit Anforderung (EA) (vgl. zusätzl. Abbildung 3.3). Architektonisch betrachtet befinden sich diese Anforderungen auf einem hohen Abstraktionsniveau (vgl. Abschnitt 3.2.1) und repäsentieren die Ansprüche an die Architektur auf einer Meta-Ebene (vgl. zusätzl. Abbildung 3.2). In [BKPS04] werden die “Identifikation architekturrelevanter Anforderungen” auch als Architekturtreiber bezeichnet und spiegeln somit den Gedanken der Kräfte (vgl. Abschnitt 3.4.2 nach dem Grundgedanken von [AIS+77]) wieder.

Bezogen auf die vorgestellten Szenarien in Abschnitt 4.1 sind folgende Architekturtreiber festzuhalten:

[1] SOTIS-City zeichnet sich dadurch aus, dass jeder Verkehrsteilnehmer ein SOTIS hat

4.2.1 Verfahren

In einem SOTIS muss es möglich sein, unterschiedliche Verfahren zu verwalten. Als mögliche Verfahren können bspw. Stauerkennung, Hinderniswarnung, Routenplanung und Notfall-Broadcast genannt werden. Zudem soll es möglich sein, dass sich die Verfahren untereinander abstimmen bzw. miteinander kommunizieren können.

EA.01 Es sollen unterschiedliche Verfahren unterstützt werden

4.2.2 Informationen

Informationen sind die Grundlagen allen Handelns in einem SOTIS. Informationen können unterschiedlichen Ursprung und unterschiedliche Qualität bzgl. des Informationsgehalts haben; bspw. können Informationen von den fahrzeugeigenen Sensoren oder von fremden Verkehrsteilnehmern kommen, die dann für den Empfänger je nach Situation unterschiedliche Bedeutung und Relevanz haben.

EA.02 Unterschiedliche Informationen sollen unterstützt werden, wie Sensordaten und Daten von anderen Fahrzeugen

4.2.3 Nebenläufigkeit

Durch die unterschiedlichen Aufgaben bzw. Verfahren, die von einem SOTIS erfüllt werden sollen, muss es möglich sein, mehrere Arbeitsabläufe gleichzeitig durchzuführen. Bspw. werden Informationen empfangen während ein Verfahren aus alten Daten einen Stau berechnet, der umgehend auf dem Display angezeigt wird. Während der selben Zeit gibt es ein weiteres Verfahren, dass z. B. den Wert eines bestimmten Sensors überwacht, somit aktiv ist, im Moment aber keine aufwendigen Berechnungen durchführt.

EA.03 Das System soll Nebenläufigkeit ermöglichen und die Abarbeitung mehrerer Verfahren und Sensorwerte unterstützen

4.2.4 Persistenz

Während des Betriebs des SOTIS werden zahlreiche Informationen von anderen empfangen und auch selbst produziert. Damit diese wertvollen Informationen nicht verloren gehen, müssen diese gespeichert, abgerufen, aktualisiert und gelöscht werden können. Weitere bisher noch nicht betrachtete Informationen stellen die Karten dar, bei denen es in den meisten Fällen ausreichend sein sollte sie abrufen zu können.

EA.04 Daten sollen gespeichert und verändert werden

4.2.5 Kommunikation

Wie in Kapitel 2 beschrieben handelt es sich bei einem SOTIS um ein kommunizierendes System (Stichwort C2CC). Die in den Verfahren produzierten Informationen sollen zum einen an andere Verkehrsteilnehmer übermittelt werden, zum anderen soll es aber auch möglich sein, Informationen von anderen zu empfangen. Die empfangenen Informationen können dann in den entsprechenden Verfahren weiterverarbeitet werden.

EA.05 Das System soll mit anderen SOTIS Systemen kommunizieren

4.2.6 Performance & Wiederverwendbarkeit

Das Einsatzgebiet eines SOTIS ist in einem Kraftfahrzeug. Durch das physikalisch begrenzte Platzangebot soll das SOTIS entweder auf der Fahrzeug eigenen Hardware[2] oder auf einem mobilen Endgerät[3] laufen und dabei mit den begrenzt vorhandenen Ressourcen funktionieren.

EA.06 Das System soll der Hardware genügen

4.3 Bewertung der Architektur-Anforderungen

EA.01 stellt eine wichtige Anforderung an die zu entwerfende SOTIS Architektur dar, da eine Voraussage bzw. endgültige Festlegung auf bestimmte Verfahren nicht möglich ist. Dies ist zum einen dadurch begründet, dass die Architektur flexibel sein soll, zum anderen zeigen einige in Abschnitt 2.4 vorgestellten Projekte einen großen Fundus an potentiell möglichen Verfahren.

EA.02 ist auch eine wichtige Anforderung, weil ohne Informationen ein SOTIS nicht existieren kann. Hier sei noch einmal auf die Bedeutung von IS (= Informations System) in SOTIS hingewiesen. Ferner sind die Verfahren auf Informationen angewiesen, so dass diese ohne Informationen nicht ihre Zuständigkeiten erfüllen können.

Die Notwendigkeit von **EA.03** und **EA.04** lassen sich dadurch ableiten, dass es mehrere Verfahren geben kann, die gleichzeitig arbeiten und dabei auf Informationen zugreifen.

Obwohl **EA.05** zu den grundlegenden Eigenschaften von einem kommunizierenden System gehört, hat es für die geplante Architektur eine niedrigere Priorität, da sich zahlreiche in Abschnitt 2.4 vorgestellten Projekte mit dieser Problematik beschäftigt haben (vgl. z. B. Abschnitte 2.4.1, 2.4.3 und 2.4.7 und [Wis07]).

EA.06 stellt, neben rein fachlichen Anforderungen auch technische Anforderungen an die Architektur, die vor dem Entwurf der Software-Architektur in dem folgenden Abschnitt weiter betrachtet werden.

[2] z. B. Fahrzeug Multimedia System
[3] z. B. Smartphone oder Navigationssystem

4.4 Analyse der technischen Anforderungen

Unter den technischen Anforderungen sind die Anforderungen an die auf der Hardware aufbauenden Software-Technologie gemeint. Verursacht durch **EA.03**, **EA.06** und die Anforderung “flexibel” in Kombination mit einem mobilen Endgerät (Navigationssystem, Smartphone) bzw. mit dem System eines Kraftfahrzeuges steht die Wahl einer Technologie aus der Java-Familie nah. Neben der hohen Akzeptanz und Verbreitung[4] sprechen auch die Portabilität, Sicherheit und Robustheit[5] für die Verwendung von Java (vgl. [BM06, KH03]). Momentan kann Java in die drei[6] Plattformen *Java Platform, Micro Edition (Java ME)*[7], *Java Platform, Standard Edition (Java SE)*[8] und *Java Platform, Enterprise Edition (Java EE)* eingeteilt werden (vgl. [Ull09]). Während Java EE überwiegend für die Entwicklung von Enterprise Anwendungen im Server Bereich verwendet wird, und basierend auf Java SE um entsprechende Pakete erweitert ist, ist Java ME eine reduzierte Java SE Version die speziell für mobile Geräte entwickelt worden ist.

4.4.1 Java Platform, Micro Edition (Java ME)

Java ME “ist eine Menge von Spezifikationen und Technologien, die speziell für mobile Endgeräte zugeschnitten sind” [BM06]. Mit “zugeschnitten” sind hier gerätespezifische Eigenschaften gemeint, d. h. unterschiedliche mobile Geräte verschiedener Hersteller stellen unterschiedliche Anforderungen an eine Java Laufzeitumgebung. In Java ME ist dieses durch die Kombination von unterschiedlichen Konfigurationen als Profile realisiert.

Die Architektur von Java ME ist in Abbildung 4.1 dargestellt und zeigt die zwei Konfigurationen Connected Device Configuration (CDC) und Connected Limited Device Configuration (CLDC) auf der untersten Ebene sowie die jeweiligen Profile auf der darüberliegenden Ebene. Auf der obersten Ebene befinden sich Optionale- und Standard-Pakete. Die CDC ist für leistungsfähigere Geräte, wie z. B. Settop-Boxen oder PDAs gedacht und verfügt über eine JVM[9]. Die CLDC ist für Mobiltelefone und ähnliche Geräte mit stärker beschränkten Ressourcen, benötigt eine Kilobyte Virtual Machine (KVM) und ist zusammen mit dem Mobile Information Device Profile (MIDP) die am meisten verbreitete Kombination. Programme, die auf der Basis der MIDP entwickelt worden sind, werden auch MIDlet genannt. Das Information Module Profile (IMP) und IMP - Next Generation (IMP NG) werden in Bereichen der Maschine-zu-Maschine-Kommunikation eingesetzt und haben weder ein Graphical User Interface (GUI) noch Sprachtelefonunter-

[4] Laut [SUN07] gibt es unter 5 Milliarden mobilen Geräten, über 2 Milliarden Mobiltelefone, die Java unterstützen

[5] durch die Verwendung der Java Virtual Machine (JVM) als *Sandbox*

[6] ohne Smart-Card

[7] früher *Java 2 Platform, Micro Edition (J2ME)* (vgl. [SUN05])

[8] bis Version 5.0 *Java 2 Platform, Standard Edition (J2SE)* benannt (vgl. [SUN06])

[9] Verglichen mit der JVM von Java SE ist diese aber im dem Leistungsumfang bzw. der Unterstützung des kompletten Java-Sprachumfangs unterlegen (vgl. [Sch07])

stützung (vgl. [Sch07]). Das Foundation Profile ist das “kleinste” Profil der CDC und verfügt wie auch das IMP über keine GUI. Das Personal Basis Profile enthält das Foundation Profile und verfügt über viele leichtgewicht Teile der Abstract Window Toolkit (AWT) Pakete. Letztlich das Personal Profile enthält auch Schwergewichtkomponenten wie z. B. `java.awt.Button` oder `java.awt.List` (vgl. [Sch07]). Eine ausführliche Übersicht der unterstützten Klassen sowie die Abhängigkeiten zwischen den verschiedenen Paketen, Profilen und Konfigurationen ist in [BM06] zu finden.

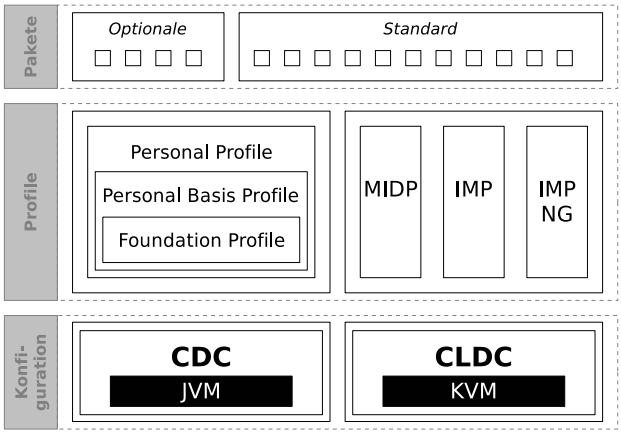

Abbildung 4.1: Java ME Architektur nach [Sch07]

4.4.2 Open Services Gateway initiative (OSGi)

OSGi steht für Open Services Gateway initiative (OSGi) und ist eine standardisierte, offene Spezifikation für ein dynamisches, plattformunabhängiges Modulsystem als Java-Framework, welches von der OSGi Allianz erarbeitet wurde und weiterentwickelt wird.

Die OSGi Allianz wurde 1999 mit dem Ziel gegründet, die Steuerung von Geräten im Heim- und Automobilbereich durch ein SOA ähnliches Konzept zu vereinfachen (vgl. [OSG09a]). Durch den großen Erfolg von OSGi ist es mittlerweile aber nicht mehr nur im Embedded Bereich, sondern auch in vielen Client- sowie Server-Applikationen anzutreffen[10] (vgl. [WHKL08]). Zu der OSGi Allianz gehören namhafte Firmen und Institute wie z. B. IBM, Sun und Automobilhersteller wie BMW (vgl. Anhang B.1 und [OSG09c]).

Von der OSGi Spezifikation gibt es eine Reihe von konkreten Implementierungen als Open Source und als kommerzielle Versionen. Eine kurze Übersicht verschafft ein Blick in Anhang B.2. Allen gemeinsam sind die beiden Hauptaufgaben:

- Module (sog. Bundles) zu verwalten und

[10] z. B. wird OSGi bei der Eclipse IDE verwendet

- die von einem Bundle angebotenen Dienste (sog. Services) als Funktionalität anderen Bundles bereitzustellen.

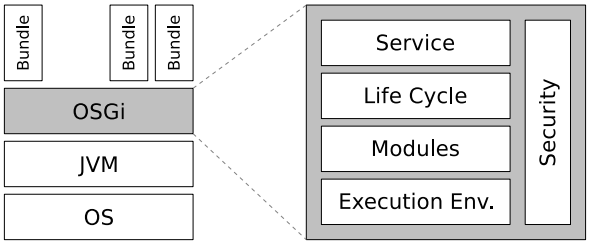

Abbildung 4.2: OSGi Architektur nach [OSG07a]

Die allgemeine Architektur von OSGi ist schematisch in Abbildung 4.2 dargestellt. Eine Implementierung benötigt nur eine auf dem Betriebssystem laufende JVM. Die unterste OSGi Schicht, das *Execution Enviorment*, stellt die Lauffähigkeit auf unterschiedlichen Java Plattformen sicher - es werden z. B. Java SE und Java ME unterstützt (vgl. [OSG09b]). Die *Modules* Schicht ist für die statischen Aspekte der Bundles zuständig und verwendet die in dem Bundle enthaltene `MAINFEST.MF` Datei, um spezifische Informationen von einem Bundle zu erhalten[11]. Technisch gesehen werden Bundles als `jar`-Dateien gespeichert und haben gegenüber normalen `jar`-Dateien den Vorteil, dass sie die Inhalte verbergen (gemäß dem Prinzip *Information-Hiding* aus Abschnitt 3.4.1). Für die dynamischen Aspekte der Bundles ist die *Life Cycle* Schicht zuständig und verwaltet die in Abbildung 4.3 gezeigten Zustände, in denen sich ein Bundle befinden kann. Die *Service* Schicht legt fest, wie Services innerhalb des Frameworks verwendet werden. Ein Service ist dabei ein einfaches Java-Objekt, das bei der Services Registry unter einem Namen angemeldet wird, und mit einem Interface für andere Services bereit steht. Zusätzlich werden von dieser Schicht selbst noch nützliche Services wie z. B. der *Service Listener* zum Überwachen von Services angeboten. Die Verwendung der *Security* Schicht ist optional und bietet die Möglichkeit, Ausführungsrechte von Bundles individuell zu konfigurieren. [WHKL08]

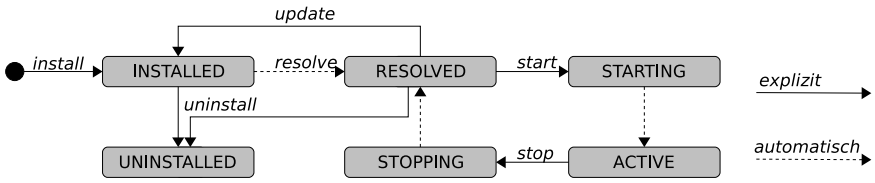

Abbildung 4.3: OSGi Lebenszyklusmodell nach [WHKL08]

[11] wie z. B. Abhängigkeiten zu anderen Bundles

4.4.3 .NET Compact Framework

Ein weiteres nennenswertes Framework - außerhalb der Java Welt - das bisher noch nicht betrachtet worden ist, ist Microsofts *.NET Compact Framework*. Dieses ist, analog zu Java ME, eine reduzierte Version des .NET Frameworks und verwendet als Ausführungsumgebung statt einer JVM eine Common Language Runtime (CLR). Plattformunabhängig können so Anwendungen auf mobilen bzw. ressourcenbeschränkten und auf Windows CE bzw. Windows Mobile basierenden Geräten ausgeführt werden (vgl. [Mic09a, Mic09d]). Die Architektur des .NET Compact Frameworks ist unter [Mic09c] zu finden. [Mic09b]

4.4.4 Android

Android ist hauptsächlich als Betriebssystem von Google für Mobile Geräte bekannt. Basierend auf einem Linux Kernel bietet Android eine Virtuelle Maschine, die sog. Dalvik VM, auf der in Java geschriebene Programme laufen. Analog zu OSGi gibt es ein Processes- und Lifecycle Management für Komponenten. Abbildung 4.4 zeigt die Architektur des Betriebsystems. Der `LocationManager` [And09c] bietet eine API zum Entwickeln von Lokations- und Karten-basierten Anwendungen. [And09a]

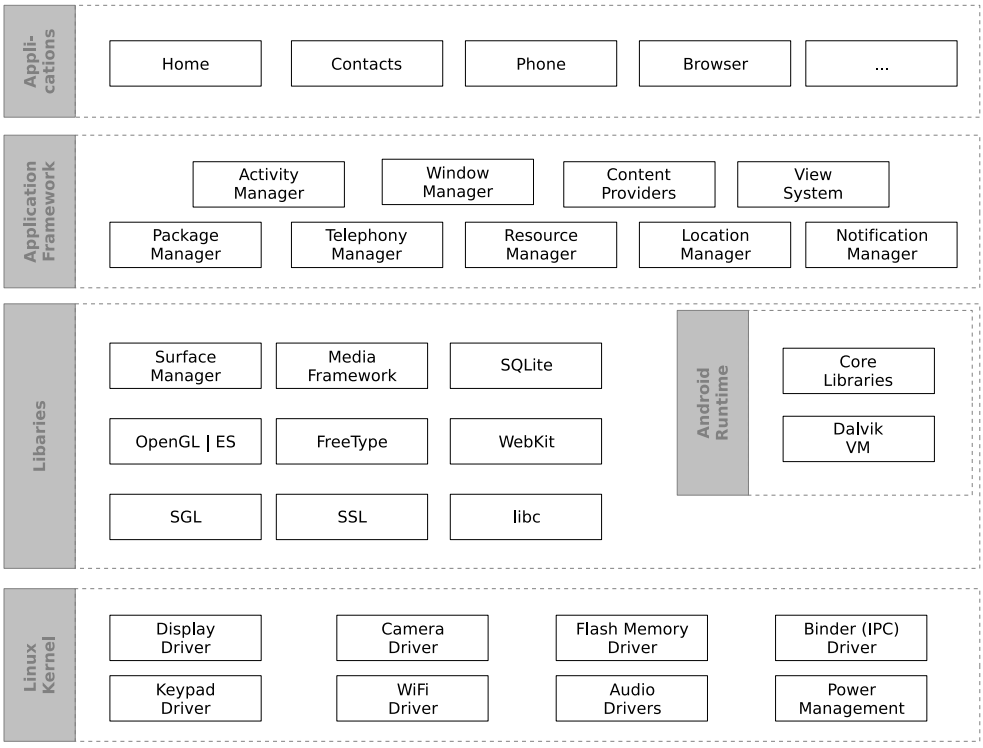

Abbildung 4.4: Android Architektur nach [And09e]

	Java ME	OSGi	.NET CF	Android
Stark verbreitet	●			
Optimiert für mobile Geräte	●	●	●	●
Plattformunabhängig	●	●	●	●
Für Fahrzeuge entwickelt		●		
Vollständige JVM bzw. CLR		●		●
Speicheroptimierung		●		
Verfolgt den SOA-Ansatz		●		

● = trifft zu

Tabelle 4.1: Vergleich von Java ME, OSGi, .NET Compact Framework und Android

4.5 Fazit

Die vier in den Abschnitten 4.4.1, 4.4.2, 4.4.3 und 4.4.4 vorgestellten Frameworks[12] sind alle für die Verwendung in mobilen Geräten geeignet. Ergänzend zu dem in Abschnitt 4.4 bereits genannten Vorteil - der weiten Verbreitung von Java - sprechen folgende Punkte für eine Verwendung von OSGi:

- Canalys[13] ermittelte für das zweite Geschäftsjahr 2007 Garmin (24,9%) und TomTom (24,8%) als die beiden weltweit Markt dominierenden Unternehmen im Bereich Navigation (vgl. [Can07]). Durch einen Hacking Angriff auf ein TomTom GO (vgl. [DK05]) wurde bekannt, dass TomTom Geräte Linux als Betriebssystem zu Grunde liegt, so das die Verwendung einer JVM möglich ist. Seitdem liegen die Sourcen unter [Tom09] mit GPL-Lizenz offen und werden von der fleißigen Open Source Gemeinde genutzt (bspw. [Ope09]). Bei Garmin kommt laut [Lin09] auch ein Linux zu Einsatz.

- Einige Automobilhersteller verwenden Linux[14] oder setzen direkt auf das OSGi Framework[15].

- Es gibt zwar Portierungen der Windows .NET Framework CLR auf Linux und MacOSX (vgl. [MP09]), inwieweit diese dann mit .NET Compact Framework Anwendungen kompatibel sind bleibt offen. Selbst die .NET Fachpresse steht der Komplexität, Technologien zusammenzuführen kritisch gegenüber, denn “je komplizierter eine Technologie ist, desto fehleranfälliger wird die Software” [Küh07] zumal portierte und ergänzte Mono CLRs [MP09] mit .NET Framework CLRs bzgl. vorhandener Pakete nicht homogen sind.

[12] bei Android inkl. Betriebssystem
[13] ein unabhängiges Analyse Institut
[14] wie z. B. Volvo (vgl. [Lin09])
[15] wie z. B. Audi (vgl. [Fri02]) und BMW (vgl. [WHKL08])

- OSGi erfüllt **EA.03**, **EA.06** und die “Flexibilität” durch (vgl. [WHKL08]):
  - Hot Deployment von Bundles und Services → dynamische Integration von Modulen (vgl. Abbildung 4.3 *update*)
  - Serviceorientiertes Programmiermodell mit Hilfe des Service Registry Konzepts → Entkopplung der Softwarekomponenten

  Zudem kann Java ME als Plattform dienen.

- Obwohl Android mit über 10.000 Programmen [And09b] sehr verbreitet zu seien scheint, gibt es laut Google’s veröffentlichter Prognose in der New York Times [And09d] bis Ende des Jahres 2009 “nur” 18 Geräte auf denen Android zum Einsatz kommen wird. Somit ist das Einsatzgebiet von Android - zumindest zurzeit - noch sehr auf bestimmte Geräte beschränkt. Zudem verwendet der `LocationManager` die kommerziellen Google Maps, so dass eine Open Source Verwendung nicht möglich ist.

Alle genannten Punkte sowie der zusammenfassende Vergleich der Eigenschaften in Tabelle 4.1 führen dazu, OSGi als Framework und Grundlage für die folgenden Kapitel auszuwählen.

Kapitel 5

Entwurf

Es gibt nie einen richtigen Weg, ein Problem zu lösen. [MPW07]

In diesem Kapitel wird die Architektur eines SOTIS Systems entworfen. Nach [VAC+05] wird mittels “Fundus an Architektur-Mitteln” (vgl. zusätzl. Kapitel 3) wie z. B. durch die Verwendung einer “bewährten Referenzarchitektur”, “eines erprobten Architektur-Musters” oder durch Anwendung “schlicht zeitloser Architektur-Prinzipien” eine Architektur entworfen. “Aus der Summe aller Entscheidungen resultiert schließlich die Architektur des Systems.” Bei dem weiteren Vorgehen werden dazu u. a. die in Abschnitt 3.2.2 vorgestellten Sichten herangezogen.

5.1 Anforderungssicht

Der Zweck der Anforderungssicht ist es die Anforderungen an die Architektur zu dokumentieren. Resultierend aus den in Abschnitt 4.2 erarbeiteten nicht-funktionalen EAen gibt es weitere funktionale Anforderungen an das System (System Anforderung (SA)) respektive deren Bausteine (Baustein Anforderung (BA)). Diese sind in Tabelle 5.1 aufgeführt und sind durch weiteres Hineindenken in die Kern-Aufgaben der Architektur entstanden.

5.2 Konzeptionelle Sicht

Das in Abschnitt 3.4.2 genannte Architekturmuster *Layers* [BMR+98] wird nun verwendet, um Struktur in das zu entwerfende SOTIS zu bringen, denn “mit Hilfe des *Layers*-Musters lassen sich Anwendungen strukturieren, die in Gruppen von Teilaufgaben zerlegbar sind und in denen diese Gruppen auf verschiedenen Abstraktionsebenen liegen” [BMR+98]. In Anlehnung an das ISO/OSI Schichtenmodell zeigt Abbildung 5.1 das

Typ	Nr.	Anforderung	Beschreibung
EA	01	Verfahren	Es sollen unterschiedliche Verfahren unterstützt werden (vgl. Abschnitt 4.2.1)
EA	02	Informationen	Unterschiedliche Informationen sollen unterstützt werden, wie Sensordaten und Daten von anderen Fahrzeugen (vgl. Abschnitt 4.2.2)
EA	03	Nebenläufigkeit	Das System soll Nebenläufigkeit ermöglichen und die Abarbeitung mehrerer Verfahren und Sensorwerte unterstützen (vgl. Abschnitt 4.2.3)
EA	04	Persistenz	Daten sollen gespeichert und verändert werden (vgl. Abschnitt 4.2.4)
EA	05	Kommunikation	Das System soll mit anderen SOTIS Systemen kommunizieren (vgl. Abschnitt 4.2.5)
EA	06	Performance & Wiederverwendbarkeit	Das System soll der Hardware genügen (vgl. Abschnitt 4.2.6)
SA	01	Local Broadcast	Daten als Local Broadcast senden (vgl. SODAD Algorithmus aus Abschnitt 2.6.2)
SA	02	Application Layer Store-and-Forward	Daten weiterleiten (vgl. SODAD Algorithmus aus Abschnitt 2.6.2)
SA	03	Datentransfer	Daten von Geräten erhalten und an diese senden
SA	04	Komponentenbasiert	Das System soll für unterschiedliche Aufgaben unterschiedliche Bausteine verwenden
SA	05	UI/GUI	Das System soll eine Benutzerschnittstelle haben
BA	01	Sensor-Daten, Wi-Fi	Es soll Bausteine geben, die die Kommunikation mit Sensor-Daten und WiFi übernehmen
BA	02	Interaktionen	Es sollen Befehle entgegengenommen werden, die dann interpretiert werden
BA	03	Ergebnisdarstellung	Es sollen Ergebnisse von Verfahren auf dem Display dargestellt werden
BA	04	Verfahren Kontrolle	Es sollen mehrere Verfahren kontrolliert (starten, stoppen) werden
BA	05	Verfahren Nebenläufigkeit	Es sollen mehrere Verfahren gleichzeitig laufen
BA	06	Verfahren Priorisierung	Die Verfahren sollen priorisiert werden
BA	07	Verfahren Standardisierung	Die Verfahren sollen alle gleich abgearbeitet werden
BA	08	Standardisierung	Es sollen einheitliche Datentypen im System verwendet werden
BA	09	Datenbank	Es soll eine Datenbank geben
BA	10	Maps	Es sollen Karten zur Verfügung stehen
BA	11	Datenverarbeitung	Ein- und ausgehende Daten sollen verarbeitet werden
BA	12	Identifikation	Nachrichten sollen eindeutig identifiziert werden

Tabelle 5.1: Anforderungen an das SOTIS

SOTIS Schichtenmodell dieses Entwurfs. Es werden dabei von oben (hohes Abstraktionsniveau) nach unten (niedriges Abstraktionsniveau) die folgenden Schichten abstrahiert:

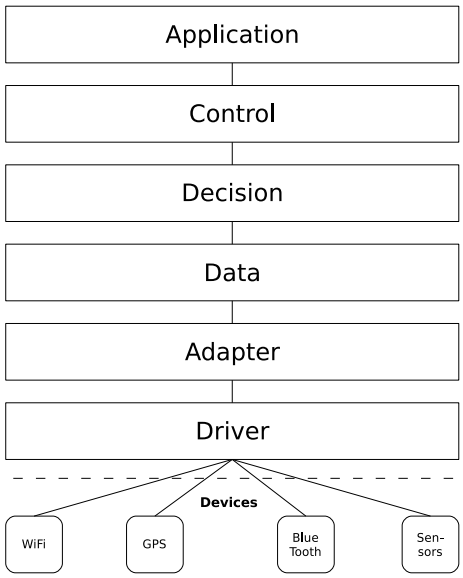

Abbildung 5.1: SOTIS Schichtenmodell

Application Schicht Eine Aufgabe der Application Schicht ist es durch die Verwendung von einem GUI bzw. UI auf einem Display, Benutzern eine Interaktionsmöglichkeit mit dem System zu geben. Bekannte Eingaben werden an die darunterliegende Schicht delegiert. Eine weitere Aufgabe ist die Darstellung von Informationen.

Control Schicht Die Control Schicht verwaltet die Verfahren des SOTIS und stellt der darüberliegenden Application Schicht Informationen sowie die Weiterverarbeitung der eingegebenen Aktionen zur Verfügung.

Decision Schicht In der Decision Schicht befinden sich unterschiedliche Verfahren, die ihre Ergebnisse der übergeordneten Control Schicht anbieten und selbst auf Ergebnisse der darunterliegenden Data Schicht zurückgreifen.

Data Schicht Die Data Schicht ist für die Verarbeitung und Weiterleitung von Daten zuständig, die empfangen, versendet oder permanent zur Verfügung stehen. Zu den Daten gehören im Wesentlichen Sensor-Daten[1], Karten-Daten und Verkehrsinformations-Daten.

[1] wie z. B. die eigene Geschwindigkeit oder die eigene GPS-Position

Adapter Schicht Die Adapterschicht dient dazu Daten von Sensoren zu holen bzw. zu bekommen und diese an die übergeordnete Data Schicht weiter zu geben. Andersherum wird die Adapter Schicht von der Data Schicht verwendet um bspw. Informationen zu versenden.

Driver Schicht In der Driver Schicht befinden sich die von den Adaptern benötigten Treiber, um Geräte (Devices) anzusteuern.

Devices Die Devices repräsentieren mögliche Geräte die von einem SOTIS verwendet werden und stellen nur indirekt eine Schicht des SOTIS dar.

5.3 Logische Sicht

Abgeleitet von den Schichten der konzeptionellen Sicht in Abschnitt 5.2 wird jeder Schicht ein eigener Baustein zugeordnet. Diese sind in Abbildung 5.2 dargestellt. Die Pfeile spiegeln dabei den Zugriff der Bausteine wieder. Dabei greift jeweils der übergeordnete Baustein auf den untergeordneten Baustein zu, also bspw. Control auf Decision. Dieses Verhalten gilt aber nur für die drei Bausteine Application, Control und Decision. Die Bausteine Data, Adapter und Driver können jeweils bidirektional aufeinander zugreifen. Ein Grund dafür sind die Bearbeiter des Chain of Responsibility-Muster (vgl. Abschnitt 5.3.4 für InputFilter und OutputFilter), die von dem Adapter Baustein sowohl Daten erhalten als auch über diesen versenden können.

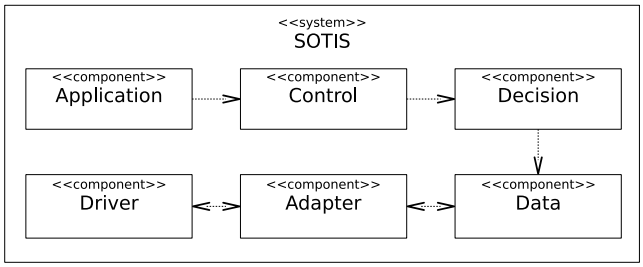

Abbildung 5.2: Gesamt-System

Basierend auf dem in Abschnitt 4.4.2 vorgestellten und ausgewählten OSGi werden innerhalb des SOTIS *Services* und *Events* als Schnittstellen zwischen den Bausteinen eingesetzt, außerhalb des SOTIS *IEEE 802.11p* (vgl. Abschnitt 2.6.3).

5.3.1 Application Baustein

Der Application Baustein enthält Klassen, die für ein UI bzw. GUI notwendig sind. Im einfachen Fall kann dieses eine Konsole sein, die Kommandos entgegen nimmt. In einer

ausgereiften Version kann es sich um ein GUI handeln, welches mittels Servlets Eingaben entgegen nimmt, durch Dialoge leitet und Karten in Form von Grafiken darstellt. Die OSGi Service Plattform bietet hierfür einen *Http Service* an [OSG07b] der zusammen mit dem Control Baustein eine Form des *Model-View-Controller* Musters [BMR+98] ermöglicht.

Der Application Baustein greift über einen Service auf den Contol (`ControlService`) Baustein zu, indem ein Kommando (`String`) übergeben wird, welches ein bestimmtes `Objekt` (dieses kann auch ein Objekt vom Typ DataType sein, vgl. Abschnitt 5.4.2) zurückliefert, das dann entsprechend dargestellt wird. Dieser Mechanismus ist in Abbildung 5.3 zu sehen und stellt den Control Service als Black-Box dar.

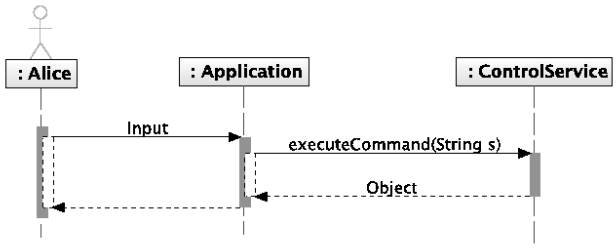

Abbildung 5.3: Sequenzdiagramm: Application Baustein

5.3.2 Control Baustein

Der Control Baustein besteht im Wesentlichen aus einem Admin Service, der für die Verwaltung von beliebig vielen SOTIS Prozessen zuständig ist, und somit die Ergebnisse von einem oder mehreren Verfahren bereitstellt. Ein SOTIS Prozess kapselt dabei ein Verfahren aus dem Decision Baustein in einem Thread und ermöglicht so eine zyklische Abfrage von Informationen. Der Admin Service selbst wird von dem bereits vorgestellten Control Service verwendet. Abbildung 5.4 zeigt die logische Struktur des Bausteins. Als Schnitt-

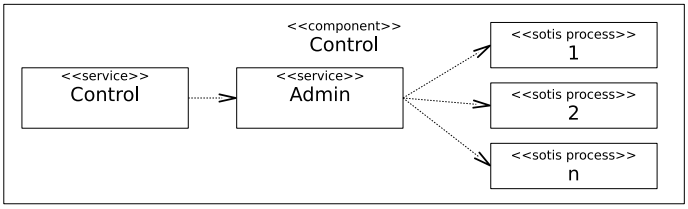

Abbildung 5.4: Baustein: Control

stelle zu dem Decision Baustein verwendet der Control Baustein einen Prozess Service, der Zugriff auf die Verfahren erlaubt. Eine weitere Aufgabe des Control Bausteins ist es die Verfahren zyklisch zu durchlaufen.

5.3.3 Decision Baustein

Hauptbestandteil des Decision Bausteins sind die Verfahren. Mit einem Verfahren kann bspw. ein Algorithmus beschrieben werden, der eine kooperative Situationsanalyse zur Stauerkennung realisiert, ein Routingalgorithmus der eine Route berechnet oder ein Verfahren, das das Weiterleiten von Verkehrsnachrichten auf der Anwendungsebene[2] sicherstellt. Ein Verfahren besteht dabei aus einer vorgegebenen Reihenfolge von Schritten die durchlaufen werden. Um eine gleiche Abarbeitung aller unterschiedlichen Verfahren zu gewährleisten wird das *Template Methode-Muster* [FF+06] als Entwurfsmuster eingesetzt. Aufbauend auf dem Wissen der vorangegangenen Kapitel werden deshalb die folgenden abstrakten Methoden festgelegt, wobei "1." nur einmal durch den Konstruktor aufgerufen wird, "2.-6." jeweils zyklisch durch eine `templateMethod()`-Methode:

1. `init()` Das Verfahren wird initialisiert. Es werden fremde Services gebunden und evtl. eigene Interfaces registriert.
2. `fetchData()` Ein Verfahren holt sich Daten von dem Data Baustein. Dieses können z. B. Sensordaten wie GPS, Nachrichten anderer Verkehrsteilnehmer oder auch bereits gespeicherte Daten aus der Datenbank sein.
3. `processData()` Anhand der geholten Daten berechnet ein Verfahren Ergebnisse, wie z. B. die Beurteilung der aktuellen Verkehrssituation wie eine Warnung für einen bestimmten Streckenabschnitt.
4. `storeData()` Anschließend werden relevante Ergebnisse in der Datenbank zwischengespeichert, bis sie bei der nächsten Iteration erneut verwendet werden.
5. `refreshData()` Die Ergebnisse, die dem Control Baustein angeboten werden, werden aktualisiert.
6. `sendData()` Sofern es notwendig ist, werden die berechneten Ergebnisse noch über den Data Baustein an andere SOTIS Teilnehmer versendet.

Nach [Wis07] (vgl. Abschnitt 2.6) und weiteren dezentralen Verkehrsinformationsprojekten (vgl. Abschnitt 2.4) werden Verfahren in unterschiedliche Klassen der Dringlichkeit eingeteilt, die wie folgt weiter verwendet werden:

Safety Safety-Verfahren haben die Aufgabe die passive Sicherheit zu verbessern

Comfort Comfort-Verfahren haben die Aufgabe Zusatzdienste anzubieten

System System-eigene-Verfahren führen Aufgaben durch, die zum Betrieb des SOTIS nötig sind

[2] Application Layer Store and Forward (vgl. Abschnitt 2.6.2)

Darüberhinaus wird die Priorisierung in diesem Entwurf durch

1. die Position eines Bearbeiters im Chain of Responsibility-Muster im Data Baustein (vgl. Abschnitt 5.3.4),
2. die Markierung jedes Datentypes mit einer entsprechenden Priorität (vgl. Abschnitt 5.4) und
3. der Möglichkeit Verfahren spezifisch zu triggern (vgl. Abschnitt 5.3.2)

sichergestellt.

5.3.4 Data Baustein

Der Data Baustein ist ein besonderer Baustein, da er unter den bisher vorgestellten als erster die Mehrfach-Anforderung schaffen muss als Schnittstelle in zwei Richtungen zu fungieren. Von “oben” greift der Decision Baustein auf ihn zu, von “unten” wird er von dem Adapter Baustein angesprochen bzw. spricht diesen selbst an. Somit ist er Vermittler für die ein- und ausgehenden Daten sowie für die Steuerung der Datenbank und Karten verantwortlich. Abbildung 5.5 zeigt die Komponenten des Data Bausteins. `Maps` stellt

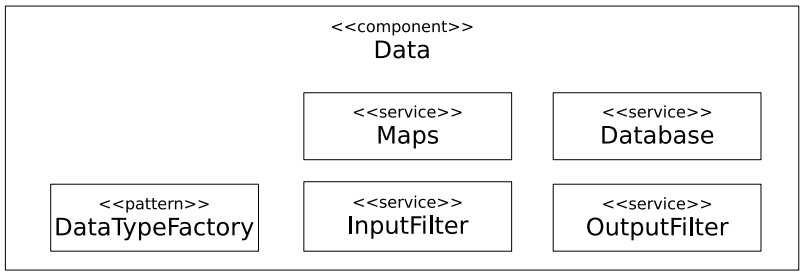

Abbildung 5.5: Baustein: Data

Karten bereit und liefert den Verfahren somit Orientierungspunkte, um Entscheidungen zu treffen und zu visualisieren. `DataBase` bietet den direkten Zugriff auf eine Datenbank. `DataTypeFactory` stellt ein einheitliches Format für unterschiedliche Daten bereit und ermöglicht so eine standardisierte und flexible Verwendung im gesamten System.[3]

Wie die Bezeichnung bereits vermuten lässt hat der `InputFilter` die Aufgabe eingehende Nachrichten zu filtern. Da in einem SOTIS nicht bekannt ist, von wem, welche Nachrichten kommen und ob diese dann überhaupt erwünscht sind, kommt hier das *Chain of Responsibility-Muster* [FF+06] als Entwurfsmuster zum Einsatz. Neben der Entkopplung der Absender und Empfänger können flexibel neue Bearbeiter für neue Nachrichtentypen in die Kette eingefügt werden. Abhängig von der Position des Bearbeiters in der Kette

[3] DataTypeFactory, Maps und DataBase werden in Abschnitt 5.4 näher betrachtet

werden die eingehenden Nachrichtentypen bearbeitet. Die - manchmal mit einem Nachteil behaftete - Eigenschaft, dass Nachrichten am Ende der Kette verworfen werden ist in diesem Fall sogar ein Vorteil, da so ungewünschten Nachrichten vorgebeugt werden kann. Abbildung 5.6 zeigt schematisch das Chain of Responsibility-Muster sowie die Verteilung der bearbeiteten Nachrichten in unterschiedliche Buffer.

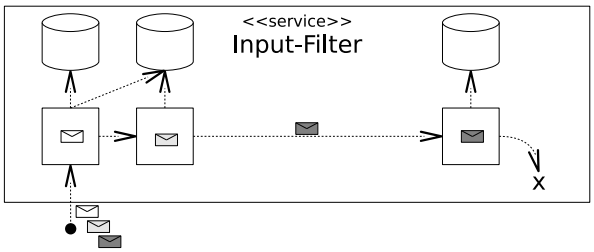

Abbildung 5.6: InputFilter

Der `OutputFilter` verhält sich analog zum InputFilter, nur das dieser für das Versenden von Nachrichten zuständig ist und entsprechende Services des Adapter Bausteins beansprucht. Das Chain of Responsibility-Muster kommt in dem OutputFilter also ein zweites Mal zum Einsatz und die Bearbeiter entscheiden individuell, wie die enthaltene Nachricht versendet werden soll; bspw. können hier zwei WiFi Bearbeiter implementiert werden, die sich um das korrekte Versenden von Local Broadcast und Application Layer Store-and-Forward Nachrichten kümmern, die sie zuvor über den Buffer von einem Verfahren erhalten haben.

5.3.5 Adapter Baustein

Der Adapter Baustein besteht aus mehreren eigenständigen Komponenten, die mit den beiden umgebenen Bausteinen (bzw. Schichten) Data und Driver kommunizieren. Diese sind jeweils in einem Thread gekapselt. Bspw. wird eine “GPS-Komponente” über den Data Baustein Werte holen und diese an den Data Baustein als Event weitergeben. Der Data Baustein selbst muss dabei aber nicht auf die “GPS-Komponente” zugreifen. Anders verhält es sich bei einer “WiFi-Komponente”. Hier werden eingehende Nachrichten an den Data Baustein weitergegeben und die “WiFi-Komponente” unterhalb des Adapter Bausteins erhält zu versendende Nachrichten. Abbildung 5.7 zeigt exemplarisch eine GPS-Komponente, die sich alle 10 Sekunden die aktuelle Position von der GPS Hardware holt, und diese als Event an den Data Baustein versendet.

5.3.6 Driver Baustein

Als Schnittstelle zur Außenwelt dient der Driver Baustein. Dieser stellt dem Adapter Baustein spezifische Klassen respektive Treiber zum Ansprechen der Hardware zur Verfü-

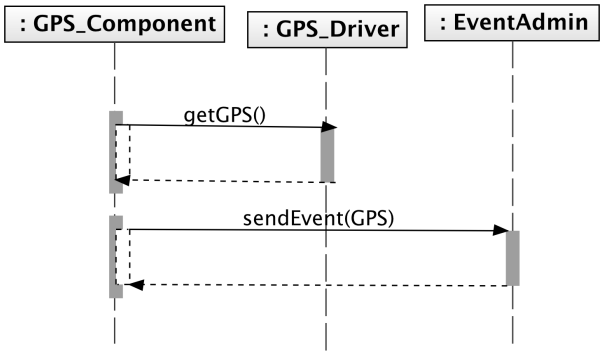

Abbildung 5.7: Sequenzdiagramm: Beispiel Adapter Baustein

gung. Neben eigenen Implementierungen kann dieser Baustein auch Bundles der OSGi Familie enthalten. In der OSGi Equinox Distribution gibt es u. a. die beiden Bundles `org.eclipse.equinox.io` und `org.eclipse.equinox.device` die eine Erkennung aller eingebauten oder angeschlossenen Hardware-Geräten sowie die Adaption der J2ME Pakete `javax.microedition.io` als Kommunikations-Infrastruktur ermöglichen sollen [Ecl09b].

5.4 Datensicht

Wie in Abschnitt 2.6.1 beschrieben sind für ein SOTIS digitale Karten[4] zur genauen Bestimmung des physikalisch existierenden Ortes eine wichtige Voraussetzung. Als weiteres stellt das Versenden und Empfangen von Nachrichten eine weitere Form intern zu bewältigender Daten bereit. Zu guter Letzt gibt es noch eine Datenbank, die für die persistente Speicherung von den Daten zuständig ist. All dieses wird in den folgenden Abschnitten weiter beschrieben.

5.4.1 Maps

Maps werden sowohl für den Endbenutzer, der diese als Graphik auf dem Display zur Orientierung sehen kann, als auch für einige der Verfahren in dem Decision Baustein, zur Berechnung von Entscheidungen oder Informationen, benötigt. Da im Rahmen dieser Arbeit nur nicht-kommerzielle Maps zur Verfügung stehen, fällt die Wahl auf OpenStreetMap (OSM) [OSM09b] die - ähnlich zu Wikipedia - von einer unermüdlichen Gemeinde von freiwilligen Mitgliedern gepflegt wird[5]. Hierdurch gibt es allerdings keine garan-

[4] im Folgenden auch Maps genannt
[5] bspw. wurden am 09.08.2009 alle Straßen von Dortmund eingepflegt [OSM09f]

Online	Offline	Eigenschaft
•		Es werden nur die Karteninformationen geladen, die benötigt werden, die Daten sind aktuell
•		Es müssen keine großen Mengen an Kartenmaterial gespeichert werden
•		Verarbeitung mit einem SAX-Parser ist möglich
	•	Es muss keine online Verbindung bestehen, die sowohl Kosten verursacht als auch die Standby-Zeit und Performance des Gerätes verringert
	•	Es besteht keine starke Abhängigkeit zu einem anderen System
	•	Zeitkritische Anfragen können garantiert werden

• = trifft zu

Tabelle 5.2: Eigenschaften: Online- und Offline-Zugriff auf Karten

tierte Qualität oder Aktualität der Rohdatenkarten, verglichen mit kommerziellen Lösungen.[6] Dafür steht OSM unter der Creative Commons Attribution-Share Alike 2.0 Lizenz [OSM09a] und kann frei[7] eingesetzt werden.

OSM bietet die Daten in Rohform (in einem XML basierten Format [OSM09c] mit der Dateiendung `.osm`) und als vorberechnete Kartenbilder (diese können als QuadTiles [OSM09e] weiterverarbeitet werden) an. Neben dem "normalen" Zugriff über das Interface der Webseite besteht die Möglichkeit eine API zur Generierung einer `.osm`-Datei zu verwenden. Die Datei kann dabei in ihrer Größe (x*y) und in ihrem Inhalt (zeige nur Autobahnen und Tankstellen an) eingeschränkt werden. Ein Beispiel dazu sowie die Koordinaten der deutschen Bundesländer und von Deutschland selbst ist in Anhang C zu finden. Für den Zugriff auf das Kartenmaterial von OSM auf einem mobilen Gerät gibt es somit zwei Möglichkeiten:

Online Benötigte Karteninformationen werden online jeweils für einen kleinen, eingeschränkten Kartenbereich über das API angefragt, heruntergeladen und portionsweise verarbeitet.

Offline Es wird einmal eine komplette Karte[8] heruntergeladen und auf dem Gerät gespeichert.

Tabelle 5.2 zeigt die Vor- und Nachteile der jeweiligen Zugriffsarten in Form von komplimentären Eigenschaften. Der Online-Zugriff wird in naher Zukunft sicherlich durch sinkende Verbindungspreise sowie fortschreitenden kabellosen Übertragungstechnologien

[6] Dieses zeigt sich z. B. bei gering besiedelten Gebieten in denen real existierende Straßen fehlen
[7] gemäß den Auflagen
[8] z. B. von Deutschland; diese ist ca. 250 MB groß

immer mehr in den Fokus rücken. Aber auch hier wird es dann, wie bei dem Offline-Zugriff, bei größeren Datenmengen zu dem Problem kommen, die Kartendateien effizient einem Verfahren anzubieten. Bspw. ist die `.osm`-Datei, die ausschließlich alle Autobahnen von Deutschland enthält 10 MB groß und besteht aus ca. 220.000 Zeilen. Die komplette Deutschlandkarte ist 250 MB groß. Spätestens hier kann ein adäquater Zugriff mittels SAX-Parser auf die Maps nicht mehr in Betracht gezogen werden.

Eine Lösung für dieses Problem bietet die Datenstruktur *R-Tree* von [Gut84]. Der R-Tree ist eine mehrdimensionale Indexstruktur für räumlich ausgedehnte, geographische Objekte (in 2D und 3D) und ist für räumliche Bereichs- und Punktabfragen geeignet. Veröffentlicht unter der LGPL-Lizenz ist die Implementierung eines R-Trees für Java unter [JSI09] zu finden.

5.4.2 DataTypeFactory und DataType

Die Anforderung unterschiedliche Informationen zu verarbeiten wird mit der `DataTypeFactory` sichergestellt. Sie befindet sich in dem Data Baustein und wird, von der konzeptionellen Sicht aus betrachtet, auch in allen darüberliegenden Bausteinen verwendet. Entstanden ist die DataTypeFactory unter Verwendung des *Factory*-Musters [FF+06] und ermöglicht durch die Factory Methode

```
abstract DataType createNewDataType(String type);        (5.1)
```

die Erstellung von Objekten, die in einer Unterklasse gekapselt sind. Dadurch wird der Code in der Superklasse von der Unterklasse entkoppelt und es können beliebige Objekte vom (abstrakten) Typ `DataType` in einer schmalen und leicht veränderbaren Schnittstelle verwendet werden. Abbildung 5.8 zeigt exemplarisch ein Klassendiagramm der DataTypeFactory mit den konkreten Datentypen Velocity (für die Geschwindigkeit), FCD (für Floating Car Data), GPS (für GPS-Daten) und AppSaF (für Application Layer Store and Forward Pakete), welche alle von DataType erben.

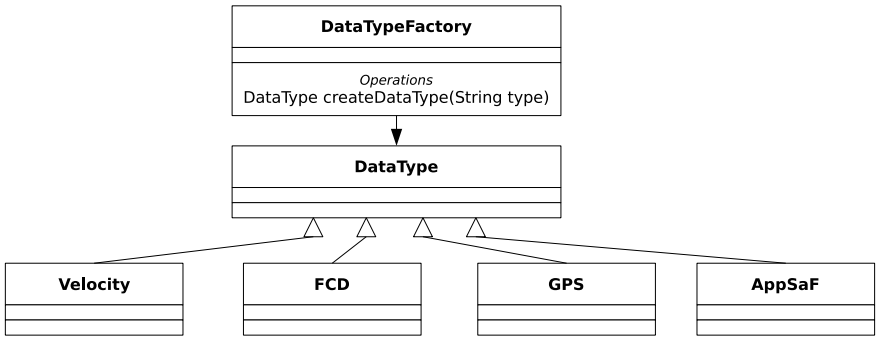

Abbildung 5.8: Klassendiagramm: DataTypeFactory

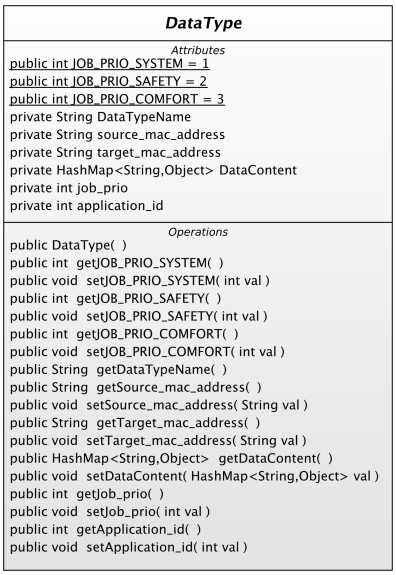

Abbildung 5.9: Klassendiagramm: DataType

5.4.3 Database

In einem SOTIS gibt es eine große Anzahl von Daten, die gespeichert werden müssen. Anders als bei “herkömmlichen” Datenbanken, wie z. B. die Daten von verschiedenen Büchern einer Bibliothek in welcher primär nach deskriptiven Stichworten (`Strings`) gesucht wird, um an eine Information zu gelangen, werden in diesem System ganz andere Kriterien in einem anderen Kontext benötigt. Im Vordergrund steht hier der Ort[9] verknüpft mit einem Zeit-/Datumstempel und einer dazugehörigen Information und ggf. die aktuelle Geschwindigkeit und die Route. Hinzukommt, dass sich durch die Bewegung der Objekte (Fahrzeuge) die Daten ständig verändern und aktualisiert werden müssen. In [HTKR05] werden u. a. kontextabhängige Anfragetransformationen mittels Selektionsbedingungen sowie komplexe Modelle zur Speicherung und Darstellung beweglicher Objekte in Datenbanken diskutiert. Eine Ausarbeitung passend für diesen Entwurf ist im Rahmen dieser Arbeit nicht vorgesehen.

5.5 Realisierungssicht

Nach [VAC+05] werden mit der Realisierungssicht die Mittel wie Programmiersprache, Frameworks, etc. ausgewählt und die Bausteine aus der logischen Sicht auf Codes verteilt. Erstes ist bereits in Kapitel 4 durch die Festlegung auf das Java basierte OSGi Framework erfolgt. Zweiteres wird in Kapitel 6 der Implementierung behandelt.

[9] meistens beschrieben durch eine GPS Koordinate

Typ	Nr.	**Anforderung** aus Abschnitt 5.1	Architektur
EA	01	Verfahren	●
EA	02	Informationen	●
EA	03	Nebenläufigkeit	●
EA	04	Persistenz	○
EA	05	Kommunikation	○
EA	06	Performance & Wiederverwendbarkeit	●
SA	01	Local Broadcast	○
SA	02	Application Layer Store-and-Forward	●
SA	03	Datentransfer	●
SA	04	Komponentenbasiert	●
SA	05	UI/GUI	●
BA	01	Sensor-Daten, WiFi	●
BA	02	Interaktionen	●
BA	03	Ergebnisdarstellung	●
BA	04	Verfahren Kontrolle	●
BA	05	Verfahren Nebenläufigkeit	●
BA	06	Verfahren Priorisierung	●
BA	07	Verfahren Standardisierung	●
BA	08	Standardisierung	●
BA	09	Datenbank	○
BA	10	Maps	●
BA	11	Datenverarbeitung	●
BA	12	Identifikation	●

● = detailliert behandelt ○ = berücksichtigt

Tabelle 5.3: Zusammenfassung des Entwurfes

5.6 Zusammenfassung

In diesem Kapitel ist, ausgehend von den Anforderungen aus Abschnitt 5.1, eine Architektur für ein SOTIS entwickelt und durch unterschiedliche Sichten dargelegt worden. So sind die einzelnen Komponenten des Systems in der logischen Sicht, basierend auf der konzeptionellen Sicht, entworfen worden. Ihre Abhängigkeiten bzgl. Datentyp und Schnittstelle sind dann in der Datensicht näher beschrieben. Tabelle 5.3 stellt die Anforderungen der Architektur gegenüber. Es ist zu erkennen, dass eine Vielzahl der Anforderungen detailliert behandelt werden konnte - einige wenige jedoch aufgrund der hohen Eigenkomplexität weitestgehend in der Architektur berücksichtigt worden sind.

Kapitel 6

Implementierung

Kunden bezahlen Sie nicht für guten Code,
sie bezahlen Sie für gute Software. [MPW07]

Basierend auf dem in Kapitel 4 ausgewählten OSGi Framework und den in Kapitel 5 erstellten Entwurf wird im Anschließenden die Implementierung beschrieben.

Für die prototypische Implementierung des Entwurfs werden insgesamt 12 OSGi Bundles verwendet. Diese sind in Tabelle 6.1 mit einer laufenden Nummer, dem symbolischen Bundle Namen, der verwendeten Version sowie dem Start Level aufgelistet.

Die Bundles 1 bis 5 entsprechen den ersten fünf Schichten des Layer Musters aus Abschnitt 5.2 respektive den Bausteinen aus Abschnitt 5.3 und werden in den folgenden Abschnitten näher erläutert. Der Driver Baustein ist nicht in der prototypischen Implementierung vorgesehen und der Adapter Baustein wird in einer angepassten Form verwendet. Die Bundles 7 bis 12 sind fertige Eclipse Equinox Bundles [Ecl09b]. Zum Finden von Programmierfehlern wurde mit Bundle Nr. 6 ein Logger implementiert, der auf dem Log Service der Plattform aufsetzt (Bundle Nr. 7). Die Bundles 8 bis 10 stellen die Unterstützung von dynamischen Services (vgl. Abschnitt 6.2) sowie die Event-Funktionalität bereit und benötigen dafür die Bundles 11 und 12. Der Start Level der Bundles legt fest, in welcher Reihenfolge die Bundles innerhalb des OSGi Frameworks gestartet werden sollen und berücksichtigt spezifische Abhängigkeiten zwischen den Bundles. Bspw. müssen die beiden Logger Bundles (Nr. 6 und 7) vor den anderen Bundles gestartet werden, damit z. B. die korrekte Initialisierung aller anschließend gestarteten Bundles geloggt werden kann.

Nr.	Bundle Name	Version	Start Level
1	unido.osgi.**application**	1.0.0	4
2	unido.osgi.**control**	1.0.0	4
3	unido.osgi.**decision**	1.0.0	4
4	unido.osgi.**data**	1.0.0	3
5	unido.osgi.**adapter**	1.0.0	4
6	unido.osgi.util.logger.consolelogger	1.0.0	1
7	org.eclipse.equinox.log	1.2.0.v20090520-1800	1
8	org.eclipse.equinox.event	1.1.100.v20090520-1800	1
9	org.eclipse.equinox.ds	1.1.0.v20090601	1
10	org.eclipse.osgi.services	3.2.0.v20090520-1800	2
11	org.eclipse.osgi.util	3.2.0.v20090520-1800	2
12	org.eclipse.equinox.util	1.0.100.v20090520-1800	2

Tabelle 6.1: Übersicht der verwendeten Bundles

Nr.	Kommando	Argument	Beschreibung
1	x	init	Initialisiert alle Verfahren
		all	Zeigt alle Informationen über die Verfahren an
2	xget	keys	Zeigt die Namen (`keys`) aller Verfahren an
3	xinfo	`<key>`	Zeigt das Ergebnis von einem Verfahren mit der Bezeichnung `<key>` an

Tabelle 6.2: Implementierte Kommandos des UI

6.1 Bundles

6.1.1 Application Bundle

Das Application Bundle befindet sich auf der höchsten Abstraktionsebene und stellt ein UI zur Eingabe von Befehlen bereit und verwendet den ControlService des Control Bundles.

unido.osgi.application.consolereader *Package*

class **Activator** Die Klasse Activator dockt an der Konsole des OSGi Frameworks an, indem ein `CommandProvider` implementiert wird, der eingegebene Befehle entgegennimmt. Über die Konsole eingegebene Befehle werden dann mit Hilfe des ControlService direkt an das Control Bundle weitergereicht. Als einfache Schnittstelle dient hier die Methode `public void executeCommand(String command, String argument);` die ein Kommando mit einem Argument als Signatur hat. Als

Rückgabewert wird hier abweichend von dem Entwurf `void` verwendet. Implementierte Kommandos sind in Tabelle 6.2 abgebildet.

6.1.2 Control Bundle

Das Control Bundle stellt anderen Bundles den Control Service zur Verfügung. Intern wird der SpAdminService verwendet. Von dem Decision Bundle wird der ProcessService benötigt.

unido.osgi.control.services *Package*

class **ControlService** Ist das Interface, das die verschiedenen Funktionalitäten des ControlServices beschreibt und enthält in der prototypischen Implementierung die Methode `executeCommand()`, die die beiden Strings Kommando und Argument interpretiert.

class **SpAdminService** Das Interface SpAdminService beschreibt die Verwaltung von sog. SOTIS Processes (SPs), die zu einer Liste hinzugefügt und entfernt werden können. Ein SOTIS Process (SP) ist dabei ein von der SotisProcess Klasse beschriebenes Verfahren des Decision Bundles, gekapselt in einem Repeater-Objekt (s. u.). Weitere Features sind das Aktivieren und Deaktivieren von SPs sowie die Festlegung und Abfrage von einem bestimmten Intervall das angibt, wie oft in der Sekunde ein SP getriggert werden soll.

unido.osgi.control.services.impl *Package*

class **ControlServiceImpl** Ist die konkrete Implementierung von ControlService aus dem Package unido.osgi.control.services und verwendet zusätzlich den SpAdminService sowie den ProcessService des Decision Bundles für die Abarbeitung aller Kommandos.

class **SpAdminServiceImpl** Die SpAdminServiceImpl ist die Implementierung des SpAdminService Interfaces.

unido.osgi.control.util *Package*

class **Repeater** Die Klasse Repeater ist eine erweiterter TimerTask Klasse und wird nur innerhalb des Control Bundles verwendet. Sie hat die Aufgabe die verschiedenen Schritte von einem SP (siehe Klasse SotisProcess Package unido.osgi.decision.pattern) nach einem bestimmten Intervall zu durchlaufen und verwendet dafür den ProcessService.

6.1.3 Decision Bundle

unido.osgi.decision.services *Package*

class **ProcessService** Das Interface ProcessService bietet als Schnittstelle nach außen die Verwaltung von SPs an. Neben den Standard-Funktionalitäten SPs hinzuzufügen, zu entfernen, die aktuelle Verfügbarkeit abzufragen und den SP zu triggern bietet der Service mit der Methode `void String getInformation(String keySP)` die Option, das aktuelle Ergebnis von einem Verfahren (mit der Identifizierung keySP) zu erhalten.

unido.osgi.decision.pattern *Package*

class **SotisProcess** Die Klasse SotisProcess implementiert das in Abschnitt 5.3.3 beschriebene Template Methode-Muster.

unido.osgi.decision.impl *Package*

class **ProcessServiceImpl** Ist die Implementierung des ProcessService.

class **AppsafSP** AppsafSP steht für *Application Layer Store-and-Forward SOTIS Process*, implementiert das SotisProcess Muster aus dem Package unido.osgi.decision.impl und repäsentiert ein Verfahren, das die Weiterleitung von Paketen auf Anwendungsebene ermöglichen soll. In der `init()`-Methode trägt sich der Prozess zuerst einmal selber in den ProcessService ein. Anschließend werden die Methoden des Musters zyklisch durchlaufen: `fetchData()` holt die für das Verfahren notwendigen Daten unter Verwendung des BufferService aus dem Data Bundle. Dabei wird die maximale Größe des Buffers berücksichtigt und geprüft, ob ein Buffer-Overflow vorliegt. In der prototypischen Implementierung ist die Größe auf 500 Einträge beschränkt. Das Handling eines potentiellen Overflows wird von dem Verfahren gehandelt, das auf den Buffer zugreift (mehr Details zum BufferService s. u.). In der `processData()`-Methode werden die abgerufenen Daten verarbeitet. Die Ergebnisse werden intern, temporär bis zum nächsten Durchlauf gespeichert (`refreshData()`) und können[1] dann mittels `sendData()`-Methode an andere Verkehrsteilnehmer gesendet werden oder mit der `storeData()`-Methode in der Datenbank gespeichert werden.

class **GpsSP** GpsSP ist ein Verfahren, das analog zum AppsafSP die Verarbeitung von GPS Daten als ein eigener SP unterstützt.

class **VelocitySP** Das Verfahren VelocitySP ist ein Verfahren, das sich um die aktuelle Geschwindigkeit kümmert (analog zu AppsafSP).

[1] in der prototypischen Implementierung nicht umgesetzt

6.1.4 Data Bundle

Das Data Bundle stellt anderen Bundles den BufferService und die DataTypeFactory samt DataType bereit und lauscht auf eingehende Events.

unido.osgi.data.lang.datatype *Package*

class **DataTypeFactory** Die Klasse DataTypeFactory implementiert den in Abschnitt 5.3.4 beschriebenen Baustein mit der Methode `public DataType createDataType(String type)`. Anhand des übergebenen `type` - der in dieser Klasse selbst definiert wird - wird ein konkretes DataType Objekt aus dem Package unido.osgi.data.lang.datatype.impl zurückgeliefert und zuvor mit einer Priorität versehen.

class **DataType** DataType bietet eine Hülle für ein beliebiges Objekt innerhalb des Systems. Neben der optionalen Verwendung der Quell- und Ziel-MAC-Adresse, der Einstufung in Prioritäten und der Zuordnung zu einer bestimmten Anwendung können beliebige Daten-Objekte in einer Key-Value HashMap hinterlegt werden.

unido.osgi.data.lang.datatype.impl *Package*

class **AppSAF** Ist die konkrete Implementierung eines Datentyps der Klasse DataType aus dem Package unido.osgi.data.lang.datatype, die bspw. von dem AppsafSP Verfahren aus dem Package unido.osgi.decision.impl verwendet werden.

class **SimpleGps** Analog zu AppSAF für einen GPS Datentyp.

class **Velocity** Analog zu AppSAF für einen Datentyp der für die Geschwindigkeit zuständig ist.

unido.osgi.data.pattern *Package*

class **ChainOfResponsibility** Die Klasse ChainOfResponsibility implementiert das in Abschnitt 5.3.4 beschriebene Chain of Responsibility Muster. Neben der Initialisierung von einem Element in der Kette mit der Methode `public ChainOfResponsibility setNext(ChainOfResponsibility cor)` werden die zu erledigenden Aufgaben über die standardisierte Interface Methode `public abstract void doJob(Event e)` an einen Bearbeiter weitergereicht. Ein Event wird dabei durch die Hilfsmethode `isPermitted()` auf Authorisierung für einen bestimmten Bearbeiter in der Kette (siehe Package unido.osgi.data.pattern.impl) überprüft.

unido.osgi.data.pattern.impl *Package*

class **AppStoreAndForwardCorImpl** Die Klasse AppStoreAndForwardCorImpl ist eine konkrete Implementierung der Klasse ChainOfResponsibility aus dem Package unido.osgi.data.pattern. Events von dem Typ *APPSTOREANDFORWARD* werden innerhalb der `doJob()`-Methode verarbeitet und über den BufferService unter Verwendung des DataTypes AppSAF in einen entsprechenden Buffer geschrieben. Events die nicht mit einem passenden Typen übereinstimmen werden mit der `doJob()`-Methode an den Nachfolger der Kette weitergegeben. Der Typ von einem Event sowie die Zuordnung der Bufferbezeichnung im BufferService wird durch die Verwendung des entsprechenden DataTypes der DataTypeFacotry in standardisierter Form sichergestellt.

class **SimpleGpsCorImpl** SimpleGpsCorImpl ist die Implementierung von einem Chain of Responsibility Element für GPS Daten die analog zu der Klasse AppStoreAndForwardCorImpl abgearbeitet werden.

class **VelocityCorImpl** Analog zu der Klasse AppStoreAndForwardCorImpl für Events bezogen auf die Geschwindigkeit.

unido.osgi.data.inputfilter *Package*

class **InGate** Die Klasse InGate ist die Schnittstelle für eingehende Events des Adapter Bundles. Um beliebige Events zu empfangen wird der `EventHandler` des OSGi Frameworks (Bundle org.eclipse.equinox.event) implementiert. Ferner initiiert diese Klasse alle Elemente des Chain of Responsibility Musters (aus Package unido.osgi.data.pattern.impl) sowie den BufferService (Package unido.osgi.data.services.impl). In dieser Implementierung erhält jeder Bearbeiter einen eigenen Buffer zum Ablegen von Daten, wobei die Buffergröße für die GPS Daten und die Geschwindigkeit jeweils 1 beträgt, da nur die aktuelle Position und Geschwindigkeit angezeigt werden. Dieser kann jedoch beliebig angepasst werden, um z. B. eine Historie der letzten 100 GPS Positionen nachzuvollziehen. Bei Eingang eines Events wird dieses an das erste Element der Kette übermittelt, welches mit der Abarbeitung beginnt.

unido.osgi.data.services *Package*

class **BufferService** Das Interface BufferService beschreibt einen BufferService und bietet die Möglichkeit beliebig viele Buffer von einem bestimmten DataType mit einer bestimmten Größe zu verwalten. Neben dem Anlegen von neuen und Entfernen von bestehenden Buffern kann die Größe nachträglich angepasst und abgefragt sowie der Zustand des Overflows variiert werden.

unido.osgi.data.services.impl *Package*

class **BufferServiceImpl** Die Klasse BufferServiceImpl ist die Implementierung des BufferServices und implementiert unter Berücksichtigung von logischen Abhängigkeiten die mit dem Interface beschriebenen Funktionen.

6.1.5 Adapter Bundle

In der prototypischen Implementierung werden Werte von Sensoren, eingehende Nachrichten von anderen SOTIS Teilnehmern und andere fahrzeugspezifische Werte wie z. B. die Geschwindigkeit simuliert.

unido.osgi.adapter.generator *Package*

class **AppSaFComponent** Die Klasse AppSaFComponent ist eine Komponente, die in beliebig festlegbaren Intervallen mit Hilfe des `EventAdmin` (Bundle org.eclipse.equinox.event) Events mit dem Typ *APPSTOREANDFORWARD* an das InGate des Decision Bundles schickt. Die Versendung des Events erfolgt asynchron durch Verwendung der `postEvent()`-Methode, die nach ihrem Aufruf direkt zum Programm zurückkehrt und wartet gegenüber der synchronen `sendEvent()`-Methode nicht auf eine Bestätigung. Dieses bietet im Wesentlichen zwei Vorteile: Einerseits ist damit das Verhalten von unregelmäßig eintreffenden Daten realistischer zu simulieren, andererseits werden Deadlocks vermieden.

class **GpsComponent** Die Klasse GpsComponent verhält sich analog zur AppSaFComponent für GPS Daten.

class **VelocityComponent** Analog zu AppSaFComponent für Geschwindigkeitsdaten.

6.2 Dynamische vs. deklarative Services

Das OSGi Framework bietet unterschiedliche Arten von Services an:

Dynamische Services Die Services werden durch eine `Activator`-Klasse, in der eine `start()` und `stop()`-Methode enthalten sind, beschrieben. Beim Starten und Stoppen des Bundles werden diese Methoden ausgeführt. Um einen eigenen Service anzubieten muss dieser bei der Registry angemeldet werden. Für das Finden von fremden Services müssen diese dann (sofern sie dynamisch verwendet werden[2]) mit Hilfe eines ServiceTrackers verwaltet werden.

[2] Dynamisch bedeutet, dass Services nach belieben vorhanden sein können oder auch nicht

Deklarative Services sind auch dynamische Services. Durch die deklarative Beschreibung mit einer `description.xml`-Datei bieten sie jedoch den Vorteil, dass sie während der Laufzeit einfacher auf Verfügbarkeit überprüft werden können. Darüber hinaus bieten deklarative Services die nachfolgenden Vorteile:

1. Sie werden schneller gestartet, es ist keine De-/Registrierung notwendig.
2. Der Speicherverbrauch ist geringer, da deklarative Services bei nicht Verwendung keine zugehörigen Klassen und Objekte laden.
3. Durch das komponentenbasierte Beschreiben im Programmiermodell[3] fallen sie weniger komplex aus.

Sowohl der Pfad zur Activator-Klasse als auch der zu einer oder mehreren description.xml-Dateien müssen in der MAINFEST.MF-Datei (vgl. Abschnitt 4.4.2) separat deklariert werden. In dieser Implementierung wird ausschließlich das Application Bundle durch keinen deklarativen Service beschrieben, da kein anderes Bundle von diesem abhändig ist und selbst keine Services anbietet. Alle anderen Bundles bzw. deren Komponenten werden jeweils durch deklarative Services[4] beschrieben. [WHKL08]

6.3 Zusammenfassung

In Tabelle 6.3 sind die System Anforderung (SA) und Baustein Anforderung (BA) den in diesem Kapitel beschriebenen Bundles gegenübergestellt. Durch die prototypische Umsetzung der Implementierung sind die Anforderungen verschieden stark in den Bundles wieder zu finden. Bspw. ist SA.02, in Abhängigkeit zu SA.03, fast vollständig umgesetzt worden und ermöglicht ein Weiterleiten von Nachrichten auf der Anwendungsebene. SA.01 wurde zwar nicht konkret implementiert, ist durch die große Ähnlichkeit zu SA.02 jedoch berücksichtigt.

SA.04 ist vollständig erfüllt und SA.05 bietet durch das integrierte UI eine Schnittstelle für einen menschlichen Benutzer an. Neun der insgesamt zwölf BA wurden detailliert bei der Programmierung beachtet. Die übrigen drei sind in dem Entwurf der Architektur aus Kapitel 5 berücksichtigt. Aufgrund ihres Aufwandes stellen sie, zusammen mit SA.03, genug Fragestellungen für mehrere weiterführende Arbeiten zur Verfügung.

Die prototypische Implementierung hat gezeigt, dass die einzelnen Komponenten ihre Aufgaben erfüllen und das System als Ganzes das gewünschte emergente Verhalten hat.

[3] Einziger Nachteil: Dieses muss erst einmal erlernt werden

[4] Die entsprechenden `.xml`-Dateien liegen jeweils vom Root-Verzeichnis des Bundles aus betrachtet im Ordner `/OSGI-INF/`

Typ	Nr.	**Anforderung** aus Abschnitt 5.1	**Application** Bundle	**Control** Bundle	**Decision** Bundle	**Data** Bundle	**Adapter** Bundle
SA	01	Local Broadcast	○	○	○	○	○
SA	02	Application Layer Store-and-Forward	●	●	●	●	●
SA	03	Datentransfer					○
SA	04	Komponentenbasiert	●	●	●	●	●
SA	05	UI/GUI	●				
BA	01	Sensor-Daten, WiFi				○	○
BA	02	Interaktionen	●				
BA	03	Ergebnisdarstellung	●				
BA	04	Verfahren Kontrolle		●			
BA	05	Verfahren Nebenläufigkeit		●	●		
BA	06	Verfahren Priorisierung		●	●		
BA	07	Verfahren Standardisierung		●	●		
BA	08	Standardisierung				●	
BA	09	Datenbank				○	
BA	10	Maps				○	
BA	11	Datenverarbeitung				●	○
BA	12	Identifikation				●	○

● = ist enthalten ○ = ist möglich

Tabelle 6.3: Zusammenfassung der Implementierung

Kapitel 7

Evaluation

Wir bauen Software wie Kathedralen.
Erst bauen, dann beten. [Gerhard Chroust]

In diesem Kapitel wird die in Kapitel 6 prototypisch implementierte Architektur auf ihr Verhalten während des Betriebs untersucht und ist somit ein Bestandteil der Verteilungssicht (vgl. Abschnitt 3.2.2) in der u. a. Performance-Aspekte betrachtet werden.

7.1 Referenzgerät

Als Referenzgerät für die Implementierung wird ein Nokia N810 Tablet verwendet, wie es in Abbildung 7.1 zu sehen ist.

Quelle: http://www.infotart.com/images/nokia-n810-press-top.jpg

Abbildung 7.1: Nokia N810 Tablet

Das Gerät ist vergleichbar mit einem modernen Smartphone, ist jedoch nicht mit Telefonfunktionalitäten ausgestattet. Auf dem Gerät ist das OSGi-Framework installiert worden.

Weitere technische Details sowie Informationen zur Installation, Konfiguration und Betrieb des Gerätes sind in Anhang D zu finden. Als zusätzliche Referenz wird in manchen Fällen ein PC[1] herangezogen.

7.2 Methodik und Objektivität der Messungen

Für die Messungen werden allgemeine Daten des Gerätes aus `/proc/meminfo` sowie programmspezifische aus `/proc/<pid>/status` über die Zeit `t` erhoben und ausgewertet. In manchen Fällen werden zusätzlich Zeitmarken verwendet, die innerhalb des Quellcodes eingefügt worden sind. Die Objektivität der Messungen unterliegt den folgenden Faktoren:

1. Die Messungen werden auf demselben System durchgeführt, auf dem das zu untersuchende Programm läuft, d. h. das insbesondere der Prozessor durch die Messvorgänge selbst beeinträchtigt wird.
2. Das Betriebssystem befindet sich im Normalbetrieb, d. h. auch andere Programme des Gerätes können unbeabsichtigt Einfluss auf die Messungen haben.
3. Die Verfahren führen keine realistischen und damit rechenintensiven Algorithmen zur Berechnung von Ergebnissen durch. Es werden keine graphisch aufwendigen Elemente wie, z. B. Karten berechnet oder auf einem GUI verwendet.
4. Die Hardware des Gerätes zur Kommunikation wie z. B. WLAN wird nicht verwendet.

7.3 Versuchsaufbau

Ein Blick auf die Struktur der Implementierung zeigt drei Punkte innerhalb des Systems, bei denen die Veränderung von Datenflüssen einen Einfluss auf das Verhalten hat:

1. **Generierung:** Die Anzahl von Informationen, die pro Sekunde eingehen (Adapter Bundle → Data Bundle)
2. **Buffer:** Die Größe der von dem BufferService verwendeten Buffern (Data Bundle)
3. **Trigger:** Die Frequenz pro Sekunde, mit denen die Verfahren getriggert werden (Decision Bundle bzw. Control Bundle)

[1] ausgestattet mit einem Intel Pentium M Prozessor 2 GHz und 1 GB Speicher

An jedem dieser drei Punkte passieren Daten bzw. sind Buffer oder Verfahren von den Typen *VELOCITY*, *GPS* und *APPSAF* zu finden, die alle untereinander abhängig sind. Ein Beispiel für APPSAF: Wird pro Sekunde 1 eingehendes APPSAF-Paket generiert (1), dann ist der APPSAF-Buffer (2) mit einer Größe von 500 nach 8 Minuten und 20 Sekunden voll. Das bedeutet, dass das APPSAF-Verfahren (3) spätenstens alle 8 Minuten getriggert werden sollte, damit es zu keinem Overflow kommt.

Im Folgenden werden für die beiden Typen VELOCITY und GPS jeweils eine Generierung alle 4 und 8 Sekunden, eine Buffergröße von jeweils 1 und ein Triggern der Verfahren alle 2 und 4 Sekunden festgelegt. Die Wahl dieser Festlegung lässt sich mit einem sinnvollen und realistischen Nutzen von Geschwindigkeits- und GPS-Daten begründen und wird für alle folgenden Test-Szenarien verwendet. Folglich werden ausschließlich die Werte für APPSAF variiert.

Ausgangssituation der nachfolgenden Szenarien: Auf dem Gerät laufen alle Standardprozesse die von dem Betriebssystem benötigt werden. Für die Ausführung der Bundles im OSGi Framework sowie die Protokollierung der Messdaten wird jeweils ein X-Terminal verwendet.

7.3.1 Versuch I - Starten des Systems

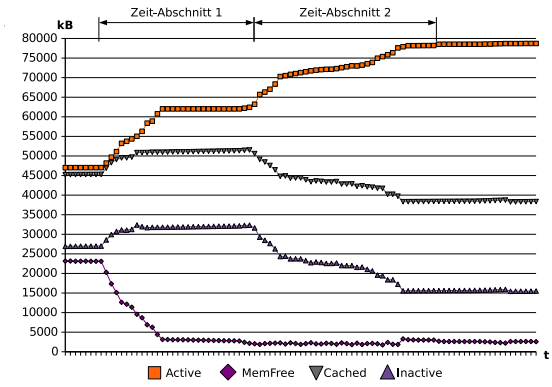

Abbildung 7.2: Speicherverbrauch beim Starten des Programmes

Abbildung 7.2 zeigt den Speicherverbrauch über eine Messdauer von 93 Sekunden, der für das Starten des OSGi Frameworks (Zeit-Abschnitt 1) und das Laden aller Bundles (Zeit-Abschnitt 2) notwendig ist. Als Quelle dienen die Daten aus /proc/meminfo. An der MemFree Kurve ist gut zu erkennen, dass OSGi direkt am Anfang den benötigten Speicher für die Bundles reserviert und diesen später, zu sehen an der Active Kurve, verwendet. Die Generierung und das Triggern der Verfahren sind beim Starten des Systems noch deaktiviert. Insgesamt braucht das Programm ca. 50 Sekunden, bis alle 12 Bundles[2] geladen

[2] vgl. Tabelle 6.1

sind und die Konsole zur Verfügung steht, um Kommandos entgegenzunehmen. Das Gerät verfügt über maximal 126828 kB Speicher (MemTotal).

7.3.2 Versuch II - Normalbetrieb

Der Speicherverbrauch von dem System im Normalbetrieb[3] ist in Abbildung 7.3 dargestellt. Links ist das Verhalten des Systems über eine Dauer von 221 Sekunden zu sehen. Zeitabschnitt 1, bezogen auf Active, zeigt einen konstanten Speicherverbrauch, nachdem das Programm gestartet und alle Bundles geladen worden sind. Die Werte stammen aus /proc/meminfo und zeigen den allgemeinen Speicherverbrauch des Gerätes. Nach Zeitabschnitt 1 wurde das Programm beendet, was durch den drastischen Abfall bzw. Anstieg von Active und FreeMem zu erkennen ist. Zeitgleich gestartet, und mit einer Messdauer von 203 Sekunden früher beendet, zeigt die Abbildung 7.3 rechts, den spezifischen Speicherverbrauch der VM, der von /proc/<pid>/status stammt. Dort ist in Zeit-Abschnitt 1 zu erkennen, dass VmData[4], VmSize[5] und VmRSS[6] im Normalbetrieb konstant bleiben. Es liegen also keine Speicherlecks vor. Weitere, nicht im Diagramm dargestellte Messungen, haben ergeben, dass der maximale Speicherverbrauch für das Programm bei 42688 kB (VmPeak) liegt.

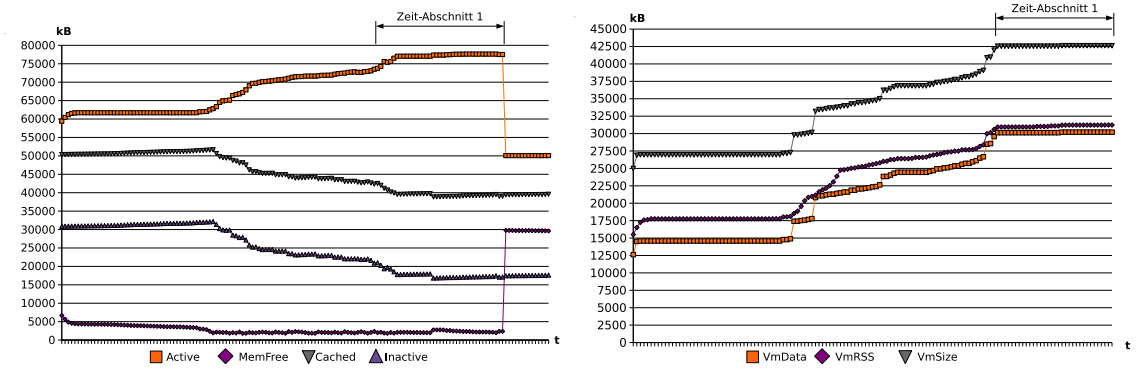

Abbildung 7.3: Speicherverbrauch im Normalbetrieb

[3] Normalbetrieb: Es wird jede Sekunde ein eingehendes APPSAF Paket generiert, das einmal pro Sekunde von dem APPSAF Verfahren aus dem 500 Elemente großen Buffer abgeholt wird. GPS und VELOCITY werden wie in Abschnitt 7.3 beschrieben verwendet

[4] VmData ist Speicher der für den Datenbereich des Prozesses genutzt wird. Statische Variablen und das Datensegment sind im Gegensatz zum Stack enthalten [LW09]

[5] VmSize ist der gesamte Speicherverbrauch des Prozesses. Enthalten sind Text- und Datensegment, Stack, statische Variablen sowie Seiten, die mit anderen Prozessen geteilt werden [LW09]

[6] VmRSS ist die Schätzung des Kernels der Resident Set Size für diesen Prozeß [LW09]

Nr.	Dauer in s	Generier. in ms	Burst	Buffergr.	Trigger in ms	N810	PC	VmPeak in kB
1	160	100	1	1.000	60.000	20-**22**	603	41780
2	287	1000	1	15	10.000	4-**5**	10	42760
3	390	1000	1	15	10.000	4-**5**	10	42404
4	183	1000	1	15	1000	1-3	2	42704
5	147	1000	1	15	500	1	1-**2**	41792
6	123	1000	1	15	100	**1**-2	**1**-2	41524
7	149	1000	1	1.000.000	100	**1**-4	1	59044
8	170	1000	1	20.000	100	1	1	46500
9	167	100	1	20.000	100	2	1	46296
10	215	10	1	20.000	100	**1**-3	10-**11**	46784
11	185	10	1	20.000	1000	**1**-2	101	42564
12	137	1000	100	20.000	1000	siehe Abb. 7.4		42756

Tabelle 7.1: Vorgabewerte und Ergebnisse

7.3.3 Versuch III - Das System bei Auslastung

In diesem Versuch wird das Verhalten der Architektur unter Auslastung näher betrachtet. Es wurden mehrere Messungen mit unterschiedlichen Konfigurationen durchgeführt, 12 davon sind in Tabelle 7.1 zusammengefasst. Alle angegebenen Werte der Spalten Generierung, Burst, Buffergröße und Trigger beziehen sich auf APPSAF; VmPeak bezieht sich auf das Gerät N810.

Die beiden Spalten N810 und PC zeigen die Anzahl, des von dem APPSAF-Verfahren erhaltenen Nachrichten an, die letztendlich auf der Konsole ausgegeben werden. Aus dem Quotienten

$$Trigger/Generierung \tag{7.1}$$

läßt sich, unter Berücksichtigung der Buffergröße, d. h. es darf zu keinem Überlauf kommen, ein idealer Wert für die enthaltenen Nachrichten bestimmen. Da die Generierung und der Trigger jeweils in eigenen Threads “gescheduled” werden, sind sowohl für das N810 als auch für den PC nicht immer ideale Werte gegeben (vgl. Messungen Nr. 1 bis 11). Messung Nr. 1 zeigt dies besonders, da das N810 - im besten Fall - mit 22 empfangenen Nachrichten deutlich unter der idealen Marke von 600 liegt. Die Buffergröße hat dabei nur einen marginalen Einfluss auf das Verhalten, wie die Messungen Nr. 6 und Nr. 7 zeigen. Insgesamt läßt sich jedoch feststellen, dass eine Triggerfrequenz die schneller als die Generierungsfrequenz ist (in Anlehnung an das Nyquist-Shannon-Abtasttheorem), ein besseres Verhalten mit sich bringt, wie Messung Nr. 8 zeigt. Hier entsprechen sowohl das N810 als auch der PC dem Ideal.

Um das Verhalten der Messungen Nr. 1 bis Nr. 11 weiter zu hinterfragen, wurde in Messung Nr. 12 der Burst von 1 auf 100 erhöht. Der Burst legt dabei fest, wie viele Nachrich-

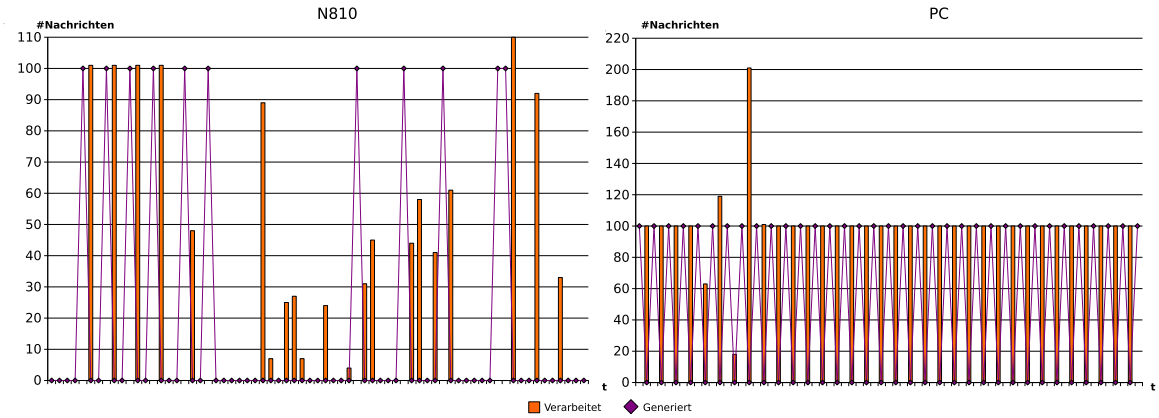

Abbildung 7.4: Diagramm: Messung 12

ten pro Generierung verwendet werden sollen, in diesem Fall also 100 Nachrichten pro Sekunde. Abbildung 7.4 links zeigt das Verhalten des N810, das anfangs mit der Verarbeitung gut zurechtkommt, in der Mitte aber keine Events mehr generiert. Zum Schluss wird die Generierung und Verarbeitung aber wieder fortgesetzt, jedoch weiterhin in einem unausgewogenen Verhältnis. Verglichen mit dem Verhalten des PCs rechts in der Abbildung stellt sich jedoch kein stabiler Zustand zwischen Generierung und Verarbeitung (Triggern) ein. Als Konsequenz für das System bedeutet dies: Nachrichten gehen zwar nicht verloren, es gibt dafür aber keine garantierten Verarbeitungszeiten.

Aus diesem Grund wurde ein Blick auf die internen Verarbeitungszeiten der APPSAF Nachrichten geworfen. Dazu wurden in dem Programmquellcode an den entsprechenden Stellen Zeitmarken mittels `System.nanoTime()`-Funktion gesetzt, die nach [KIT09] "derzeit für Performance-Messungen die beste Wahl" ist. In Tabelle 7.2 sind die Ergebnisse der Messungen zusammengefasst. Die Zeilen Nr. 1 bis 3 zeigen, wie lange sich eine APPSAF Nachricht in der Komponente aufhält. Die Zeilen 3.a bis 3.e zeigen die Zeit die für die Methoden des Template Method-Muster (vgl. Abschnitt 5.3.3) notwendig sind. Der Vergleich zwischen der Summe (3.f) der einzelnen Methoden (3.a bis 3.e) der Decision Komponente und der Gesamtzeit der Decision Komponente (3), zeigt die Lücken die durch den Scheduler entstehen. Die Zeilen 4.a, 4.b, 5.a und 5.b zeigen die Zeiten, die von den Nachrichten benötigt werden, um von einer Komponente in die nächste zu wechseln. Je nach Überlappung von Generierung und Trigger kann dieses bei dem Wechseln von dem Adapter- zur Data-Komponente bei dem N810 schon mehrere Sekunden dauern (4.a und 5.a). Der PC ist deutlich schneller. Der Wechsel von der Data- zur Decision-Komponente hingegen wird beim N810 bei Verwendung eines adäquaten Generierung-Trigger-Verhältnisses, wie es mit Konfig 2 vorliegt, in durchschnittlich 407 ms geleistet und bestätigt die in Messung Nr. 12 festgestellte Auslastung durch den Single Burst.

	Alle Zahlenwerte, außer der Buffergröße, in ms	N810 Messung 1	N810 Messung 2	N810 Messung 3	N810 Ø	PC Messung
	Konfig 1: Generierung: 5000 / Buffergröße: 20000 / Verfahren-Trigger: 5000					
1	Adapter	3785,1	1689,9	1522,1	2332,4	1,1155
2	Data	1063,8	2231,5	3054,1	2116,4	0,3280
3	Decision	17,790	13,770	17,670	16,410	7,1763
3.a	Decision (fetchData)	9,9203	5,7400	10,130	8,5967	6,6109
3.b	Decision (processData)	1,0698	1,1000	1,0698	1,0799	0,0908
3.c	Decision (storeData)	0,8602	0,8599	0,8599	0,8600	0,0662
3.d	Decision (refreshData)	0,8899	0,8899	0,9700	0,9166	0,0662
3.e	Decision (sendData)	0,8200	0,8602	0,8499	0,8434	0,0662
3.f	Decision (Summe)	13,560	9,4500	13,880	12,297	6,9003
4.a	Adapter → Data	2669,7	4831,7	5184,8	4228,7	0,9163
4.b	Data → Decision	3825,7	3208,3	1825,6	2953,2	26,880
	Konfig 2: Generierung: 5000 / Buffergröße: 20000 / Verfahren-Trigger: 1000					
5.a	Adapter → Data	8227,8	1,4351	5635,0	4621,4	0,2403
5.b	Data → Decision	430,63	51,330	741,27	407,74	239,31

Tabelle 7.2: Laufzeiten von APPSAF durch die Bundles

7.4 Zusammenfassung

Die Evaluation des Systems auf Basis der entworfenen Architektur hat gezeigt, dass das System nach durchschnittlich 50 Sekunden einsatzbereit ist. Der Speicherverbrauch im Normalbetrieb ist angemessen und lässt genug (Speicher-)Platz für zukünftige Erweiterungen.

In Versuch III ist, wie angenommen, der Flaschenhals des Systems entdeckt worden: Das N810 hat bei hoher Last und schlechter Konfiguration Schwierigkeiten damit, regelmäßige Events der OSGi Plattform zu erzeugen und entsprechend schnell abzuarbeiten. Bei einer guten Konfiguration, wie sie in Messung Nr. 8 (vgl. Tabelle 7.1) vorliegt, liefert das System aber auch unter Last ein ausreichend gutes Ergebnis. Desweiteren wurde durch 5.b (vgl. Tabelle 7.2) eine gute Weiterverarbeitung der Nachrichten - nach dem Flaschenhals - nachgewiesen und somit ein performantes Zusammenspiel der Komponenten und Services gezeigt. Neben einer guten Konfiguration besteht die Möglichkeit, die verwendeten Events der Adapter-Komponente durch neue Services der Data-Komponente zu ersetzen, um den potentiellen Flaschenhals zu beheben.

Kapitel 8

Fazit und Ausblick

8.1 Fazit

Angefangen mit den in Kapitel 2 vorgestellten Verkehrsprojekten aus den Bereichen der Fahrzeug-zu-Fahrzeug Kommunikation und der dezentralen Verkehrsinformationssytemen, boten diese einen immens großen Fundus potentieller Anforderungen und Funktionalitäten an, die ein System erfüllen könnte oder auch sollte. Eine große Herausforderung war es, ein ausgewogenes Gleichgewicht zwischen *zu vielen* und *zu wenigen* Anforderungen zu finden - zumal *alle* Anforderungen unterschiedliche Wichtigkeit für ein SOTIS haben.

Um Herr der sinnvollen und wichtigen Anforderungen zu bleiben, wurde mit dem Kapitel 3 über die Architektur in der Informatik eine Grundlage geschaffen, mit dessen Hilfe ein strukturiertes und erprobtes Vorgehen zum Entwerfen einer Architektur möglich ist. Daraus resultierten in Kapitel 4, unter Verwendung der *Warum*-Dimension, dann die wichtigsten Anforderungen an die Architektur. Wegen der Services, der modularen, flexiblen Gestaltung der Komponenten des Systems in Bundles und der großen Akzeptanz und Kompatibilität, wurde in diesem Kapitel das OSGi Framework als Basis für die Architektur ausgewählt.

Für den eigentlichen Entwurf der Architektur in Kapitel 5 wurden die in Kapitel 3 vorgestellten Sichten der *Wo*-Dimension verwendet, um die unterschiedlichen Aspekte der Ebenen des Systems zu berücksichtigen. Mit der *Womit*-Dimension wurde die Architektur dann in dem Grobentwurf mit Hilfe des Architekturmusters *Layers* in unterschiedliche Schichten eingeteilt. Dieses bietet zurzeit eine klare Trennung der Zuständigkeiten und ermöglicht zukünftig, leicht Erweiterungen und Anpassungen an dem architektonischen Entwurf durchzuführen.

Durch Anwendung der verschiedenen Sichten wurden die unterschiedlichen Seiten der Architektur hinterfragt und kontextbezogen dargelegt. Unter Verwendung der Sichten auf die Schichten sowie durch den Einsatz von Entwurfsmustern, wie das Chain of Responsibility-Muster, Template Methode-Muster und Factory-Muster, wurden weitere Verfeinerungen

Meilenstein	Erreicht?
SOTIS & Verkehrs-IS	√
Theorie über Architektur	√
Analyse	√
Entwurf	√
Implementierung	√
Evaluation	√

Tabelle 8.1: Meilensteine der Diplomarbeit

vorgenommen. Zusätzlich wurden bei dem Entwurf die Prinzipien der losen Kopplung, der hohen Kohäsion, des Information-Hiding, der Abstraktion und der Modularität eingehalten und in Kapitel 6 bei der Implementierung umgesetzt. So wurde das *Veränderliche* in eigenen Datentypen gekapselt und beim Programmieren gegen die Schnittstelle programmiert, so dass durch die Einhaltung der Prinzipien ein erweiterbares und flexibles System entstanden ist.

Grob betrachtet sollte sich diese Arbeit an dem Wasserfallmodell orientieren. Durch die vorgenommene Verfeinerung des Entwurfs, die als Erfahrung während der Implementierung entstanden ist, wurde der Entwurf iterativ (vergleichbar als Teilprozess des Spiralmodells) immer weiter verfeinert.

In Kapitel 7 wurde die prototypisch implementierte Architektur dann auf einem mobilen Nokia N810 Tablet auf korrekte Funktionsweise im normalen und belasteten Betrieb untersucht. Das Ergebnis: Im Normalbetrieb funktioniert die Implementierung wie geplant, in Last-Szenarien zeigt sie (erwartete) Schwächen.

Tabelle 8.1 zeigt abschließend die Bewertung der erreichten Ziele dieser Diplomarbeit.

Mein persönliches Resümee

Der Entwurf und die Implementierung einer Architektur erfordern viel Erfahrung in Theorie und Praxis sowie Geschick diese in einer angemessenen Form miteinander zu verwenden.

8.2 Ausblick

Die entworfene SOTIS Architektur bietet die Möglichkeit für zukünftige Anforderungen modular erweitert und weiterentwickelt zu werden. Besonders die prototypische Implementierung sollte unter Verwendung von realen GPS- und Sensor-Daten sowie die Kommunikation via WiFi weiter getestet werden. Zudem bietet die Architektur die Möglichkeit mehrere Verfahren gleichzeitig einzubinden. Hier könnte ein, parallel und unabhängig zu dieser Arbeit entstandene, Verfahren zur kooperativen Situationsanalyse [Hof09] eingebunden und getestet werden.

Um weitere Features des OSGi Frameworks zu nutzen, sollten die Verfahren als Fragment-Bundles verwendet werden. Desweiteren ist die effiziente Darstellung von Kartendaten und die Verwendung des Application Admin Service zu prüfen.

Die Sicherheit in mobilen Adhoc Netzen wurden in dieser Arbeit nicht betrachtet. [Mag04] bietet einen interessanten Ansatz um diese zu berücksichtigen.

Als Weiterentwicklung des verwendeten OSGi Frameworks sollte noch das, sich zur Zeit in der Inkubations-Phase befindliche, Eclipse Projekt Swordfish [Ecl09c] erwähnt werden. Es basiert auf Equinox und ist ein Open Source SOA Runtime Framework. Mit einem weiteren Projekt namens Pulsar [Ecl09a] stellte Eclipse Mitte dieses Jahres eine neue Plattform zur Entwicklung mobiler Anwendungen bereit. Involvierte Projektpartner sind u. a. Motorola, Nokia, Genuitec, RIM, Sony Ericsson und IBM. Ein Ziel ist es, die Interoperabilität der Java ME Distributionen zu vereinheitlichen.

Was bleibt, ist die Veränderung; was sich verändert, bleibt.
[Michael Richter]

Anhang A

Sensoren in Fahrzeugen

Motormanagement

Heißfilm-Luftmassenmesser Erfassen der Luftmasse im Saugrohr

Drehzahlsensor Berührungsloses Erfassen des Kurbelwellen-Drehwinkels und der Kurbelwellen-Drehzahl

Phasensensor Berührungslose Erfassung des Nockenwellen-Drehwinkels

Drucksensoren Barometrische Druckmessung

Klopfsensor Erfassen von Körperschallschwingungen (Klopfen). Eine klopfende Verbrennung kann den Motor schädigen. Die Daten des Klopfsensors ermöglichen ein Gegensteuern.

Temperatursensor Erfassen der Temperatur von Kühlmittel, Kraftstoff und Luft

Fahrpedalmodul Erfassen des Fahrerwunschs (Drehmoment-Anforderung)

Abgasmanagement

Abgassensor Messung der Sauerstoffkonzentration im Abgas. Diese Messgröße ist die Grundlage für den geregelten Dreiwege-Katalysator (Benzinmotor) und für die Lambda-Control-Funktion (Dieselmotor)

Differenzdrucksensor Erfassen des Beladungszustands des Dieselpartikelfilters

Getriebetechnik

Drehzahlsensor Getriebe Berührungsloses Erfassen von Eingangs- und Ausgangsdrehzahl bei Automatikgetrieben

Lenkung

Lenkwinkelsensor Berührungsloses Erfassen des Lenkwinkels

Drehmomentsensor für Lenksysteme Berührungsloses Erfassen des Drehmoments

Aktive Fahrsicherheit

Raddrehzahlsensor Berührungsloses Erfassen der Raddrehzahl; Zusatzfunktionen für neue Systeme integrierbar, z. B. Drehrichtungserkennung, Stillstandserkennung und Luftspaltdetektion

Lenkwinkelsensor Berührungsloses Erfassen des Lenkwinkels

Drehratensensor Erfassen der Drehbewegung des Fahrzeugs um seine Hochachse

Regensensor

Passive Fahrsicherheit

Drehratensensor Erfassen von Drehbewegungen um die Längsachse des Fahrzeugs

Beschleunigungssensor Erfassen von Beschleunigungen in ein oder zwei Achsen

Peripherer Drucksensor PPS Messung der dynamischen Druckänderung durch Türdeformation im Seitencrash und zusätzlicher Sensierung des Absolutdrucks

Klimaüberwachung

Climate Control Sensor Bestimmung der CO_2 Konzentration im Fahrzeuginnenraum

Energiemanagement

Elektronischer Batterie-Sensor (EBS) Erfassung der Batteriegrößen Strom, Spannung und Temperatur; Informationen über den aktuellen Zustand der Batterie und über das zukünftig erwartete elektrische Verhalten

Anhang B

OSGi

B.1 OSGi Mitglieder Firmen

1. Alpine Electronics Europe Gmbh
2. Aplix Corporation
3. Belgacom
4. BMW Group
5. Cablevision Systems
6. Computer Associates
7. Deutsche Telekom AG
8. Echelon Corporation
9. Electricité de France (EDF)
10. Ericsson Mobile Platforms B
11. Esmertec
12. Espial Group, Inc.
13. ETRI Electronics and Telecommunications Research Institute
14. France Telecom
15. Gatespace Telematics AB
16. Gemplus

17. Harman/Becker Automotive Systems GmbH
18. Hitachi, Ltd.
19. IBM Corporation
20. Industrial Technology Research Institute
21. Insignia Solutions
22. Intel Corporation
23. KDDI R&D Laboratories, Inc.
24. KT Corporation
25. Mitsubishi Electric Corporation
26. Motorola, Inc.
27. NEC Corporation
28. Nokia Corporation
29. NTT
30. Oracle Corporation
31. Panasonic Technologies, Inc.
32. Philips Consumer Electronics
33. ProSyst Software GmbH
34. Robert Bosch Gmbh
35. Samsung Electronics Co., Ltd.
36. SavaJe Technologies, Inc.
37. Sharp Corporation
38. Siemens AG
39. Sun Microsystems, Inc.
40. Telcordia Technologies, Inc.
41. Telefonica I+D
42. TeliaSonera

43. Toshiba Corporation
44. Vodafone Group Services Limited

Quelle: [OSG07b]

B.2 OSGi Implementierungen

Übersicht bekannter OSGi Implementierungen (unvollständige Liste):

- Open Source
  - Eclipse Equinox
    `http://www.eclipse.org/equinox/`
  - Apache Felix
    `http://cwiki.apache.org/FELIX/index.html`
  - Knopflerfish
    `http://www.knopflerfish.org/`
  - ProSyst mBedded Server Equinox Edition
    `http://www.prosyst.com/products/osgi_se_equi_ed.html`
- Kommerzielle
  - ProSyst
    `http://www.prosyst.com/`
  - Knopflerfish Pro
    `http://www.gatespacetelematics.com/`

Anhang C

OpenStreetMap

In OSM werden in einem umschliessenden Rechteck, der sog. Bounding Box (bbox) Longitude ($long_{von}$ und $long_{bis}$) und Latitude (lat_{von} und lat_{bis}) Werte angegeben. Die Bounding Box für Deutschland ist z. B. `5.185546875,46.845703125,15.46875,55.634765625`. Listing C.1 zeigt den `wget`-Befehl zum Speichern aller Autobahnen in Nordrhein-Westfalen und speichert diese in einer `.osm` Datei. In den Tabellen C.1 und C.2 sind die bbox-Werte für die Bundesländer in unterschiedlichen Formaten dargestellt. [OSM09d]

```
wget –O NRW_Autobahnen.osm
    http://xapi.openstreetmap.org/api/0.5/
    [highway=motorway] [bbox=5.8659988131, 50.3226989435,
    9.4476584861, 52.5310351488]
```

Listing C.1: Befehl zum Downloaden der NRW Autobahnen als OSM

Bundesland	**Koordinaten** (`bbox=`)
Baden-Württemberg	6.7.5113934084,47.5338000528,10.4918239143,49.7913749328
Bayern	8.9771580802,47.2703623267,13.8350427083,50.5644529365
Berlin	13.0882097323,52.3418234221,13.7606105539,52.6697240587
Brandenburg	11.2681664447,51.3606627053,14.7647105012,53.5579500214
Bremen	8.4813576818,53.0103701114,8.9830477728,53.6061664164
Hamburg	53.3949251389,8.4213643278,10.3242585128,53.9644376366
Hessen	7.7731704009,49.3948229196,10.2340156149,51.6540496066
Mecklenburg-Vorpommern	10.5932460856,53.1158637944,14.4122799503,54.6849886830
Niedersachsen	6.6545841239,51.2954150799,11.59769814,53.8941514415
Nordrhein-Westfalen	5.8659988131,50.3226989435,9.4476584861,52.5310351488
Rheinland-Pfalz	6.1173598760,48.9662745077,8.5084754437,50.9404435711
Saarland	6.3584695643,49.1130992988,7.4034901078,49.6393467247
Sachsen	11.8723081683,50.1715419914,15.0377433357,51.6831408995
Sachsen-Anhalt	10.5614755400,50.9379979829,13.1865600846,53.0421316033
Schleswig-Holstein	7.8685145620,53.3590675115,11.3132037822,55.0573747014
Thüringen	9.8778443239,50.2042330625,12.6531964048,51.6490678544

Tabelle C.1: OpenStreetMap: Koordinaten der Bundesländer

Bundesland	Norden	Westen	Süden	Osten
Baden-Württemberg	49.7913749328	7.5113934084	47.5338000528	10.4918239143
Bayern	50.5644529365	8.9771580802	47.2703623267	13.8350427083
Berlin	52.6697240587	13.0882097323	52.3418234221	13.7606105539
Brandenburg	53.5579500214	11.2681664447	51.3606627053	14.7647105012
Bremen	53.6061664164	8.4813576818	53.0103701114	8.9830477728
Hamburg	53.9644376366	8.4213643278	53.3949251389	10.3242585128
Hessen	51.6540496066	7.7731704009	49.3948229196	10.2340156149
Mecklenburg-Vorpommern	54.6849886830	10.5932460856	53.1158637944	14.4122799503
Niedersachsen	53.8941514415	6.6545841239	51.2954150799	11.59769814
Nordrhein-Westfalen	52.5310351488	5.8659988131	50.3226989435	9.4476584861
Rheinland-Pfalz	50.9404435711	6.1173598760	48.9662745077	8.5084754437
Saarland	49.6393467247	6.3584695643	49.1130992988	7.4034901078
Sachsen	51.6831408995	11.8723081683	50.1715419914	15.0377433357
Sachsen-Anhalt	53.0421316033	10.5614755400	50.9379979829	13.1865600846
Schleswig-Holstein	55.0573747014	7.8685145620	53.3590675115	11.3132037822
Thüringen	51.6490678544	9.8778443239	50.2042330625	12.6531964048

Tabelle C.2: OpenStreetMap: Koordinaten der Bundesländer in WGS84 und DecDeg

Anhang D

N810

D.1 Geräteeigenschaften

Das Nokia N810 verfügt über WLAN IEEE 802.11b/g, Bluetooth 2.0, 2 GByte interner Speicher für Nutzerdaten und ein hochauflösendes Touchscreen Display (800 x 480 Pixel) mit 10,5 cm Diagonale und 65.536 Farben. Es ist erweiterbar und kompatibel mit miniSD- und microSD-Speicherkarten mit einer Größe von bis zu 8 GByte. Für die Rechenleistung sorgt ein TI OMAP 2420 Processor mit 400 MHz der bei kontinuierlicher Nutzung, d. h. bei eingeschaltetem Display und aktiviertem WLAN bis zu 4 Stunden durchhält. Die Stand-by-Zeit beträgt bis zu 14 Tage. Als Betriebssystem wird Maemo[Mae09] ein Open-Source Linux Klon verwendet. [Nok09a]

D.2 Installation und Konfiguration

D.2.1 Red pill mode

Der Red pill mode ist ein spezieller Modus des N810, in welchem erweiterte Einstellungen möglich sind. Achtung: Der Red pill mode deaktiviert zahlreiche Sicherheitseinstellungen des Applicationmanagers und sollte nur von Administratoren verwendet werden.

Aktivieren des Red pill mode:

1. *Settings* → *Application Manager*
2. Im Menü (oben links): *Tools* → *Application catalogue...*
3. *New* → `matrix` in die Webadresse eingeben (ohne http:// etc.)
4. *Cancel* auswählen

5. *Red* auswählen

Zum Deaktivieren des Red pill mode Schritte 1 bis 4 erneut durchführen und in Schritt 5 *Blue* auswählen. [N8108b]

D.2.2 Nützliche Programme

In Vorbereitung auf die Installation von Programmen wie Java oder auch OSGi sind *openssh*, *unzip* und *wget* nützliche Programme, die vorab installiert werden sollten. Von *openssh* sollte die Client- und Server-Version installiert werden. Nachfolgend sind die Schritte aufgezeigt, die für eine Installation durchgeführt wurden:

1. *Settings* → *Application Manager*
2. Im Menü (oben links): *Tools* → *Application catalogue...*
3. *New* →

   manemo Extras bei Catalog name,

   `http://repository.maemo.org/extras/` bei Web address und

   free non-free bei Components eintragen
4. *OK auswählen*
5. Zurück zur Übersicht *Application manager* → *Browse installable applications* und folgende Progamme auswählen und installieren:

   Programm (1/3): **openssh** Eigenes Passwort festlegen, hier PW=`ssh`

   Programm (2/3): **wget**

   Programm (3/3): **unzip**

D.2.3 `su` und `sudo` für `user` aktivieren

Zum Ausführen von einigen Skripten und Programmen in der Shell sind `sudo`-Rechte notwendig.

1. Mit *ssh* auf das Gerät zugreifen
2. Als `root` mit entsprechendem Passwort (wie oben festgelegt) einloggen
3. Anschließend mit `passwd user` ein neues Passwort für den Benutzer `user` anlegen
4. `echo "user ALL = PASSWD: /bin/su" >> /etc/sudoers` eingeben, um dem `user` Benutzer `su` zu ermöglichen

5. Optional kann der remote Login für `root` durch Eingabe von `passwd -l root` deaktiviert werden

Hinweis: Das neue Passwort für `user` ist: `nokia` [N8108a]

D.2.4 OSGi, Java und Co.

Das in Kapitel 6, Tabelle 6.1 genannte OSGi Framework Bundle wird auf dem Nokia N810 zusammen mit Java 1.5.0 verwendet. Java selbst verwendet CACAO 0.99.4 [CAC09b] welches unter der Cacao-Lizenz [CAC09a] steht. CACAO selbst benötigt wiederum GNU Classpath [GNU09a] das unter der GNU General Public License [GNU09b] steht.

D.3 Betrieb

D.3.1 Konfigurationsdatei

Die Konfigurationsdatei `config.ini` wird zum automatischen Laden und Starten aller Bundles in der korrekten Startreihenfolge benötigt.

```
osgi.bundles=\
file:/home/user/MyDocs/my/equinox/framework/org.eclipse.equinox.log_1.2.0.v20090520-1800.jar@1:start,\
file:/home/user/MyDocs/my/equinox/framework/org.eclipse.equinox.event_1.1.100.v20090520-1800.jar@1:start,\
file:/home/user/MyDocs/my/equinox/framework/org.eclipse.equinox.ds_1.1.0.v20090601.jar@1:start,\
file:/home/user/MyDocs/my/equinox/framework/org.eclipse.osgi.services_3.2.0.v20090520-1800.jar@2:start,\
file:/home/user/MyDocs/my/equinox/framework/org.eclipse.osgi.util_3.2.0.v20090520-1800.jar@2:start,\
file:/home/user/MyDocs/my/equinox/framework/org.eclipse.equinox.util_1.0.100.v20090520-1800.jar@2:start,\
file:/home/user/MyDocs/my/equinox/framework/unido.osgi.util.logger.consolelogger_1.0.0.jar@1:start,\
file:/home/user/MyDocs/my/equinox/framework/unido.osgi.data_1.0.0.jar@3:start,\
file:/home/user/MyDocs/my/equinox/framework/unido.osgi.adapter_1.0.0.jar@4:start,\
file:/home/user/MyDocs/my/equinox/framework/unido.osgi.application_1.0.0.jar@4:start,\
file:/home/user/MyDocs/my/equinox/framework/unido.osgi.decision_1.0.0.jar@4:start,\
file:/home/user/MyDocs/my/equinox/framework/unido.osgi.control_1.0.0.jar@4:start,\
eclipse.ignoreApp=true
```

Listing D.1: config.ini

D.3.2 Starten der SOTIS Anwendung

Die SOTIS Anwendung bzw. die prototypische Implementierung wird durch Aufruf der folgenden Zeile gestartet, wobei die config.ini in dem Unterordner config liegen muss:

```
java -jar org.eclipse.osgi_3.5.0.v20090520.jar
  -configuration
  /home/user/MyDocs/my/equinox/framework/config -console
  -clean
```

Listing D.2: Befehl zum Starten der Anwendung

Anhang E

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ACC	Adaptive Cruise Control
ADAC	Allgemeine Deutsche Automobil-Club
AHS	Advanced Cruise-Assist Highway System
AHSRA	Advanced Cruise-Assist Highway System Research Association
AKD	Aktivitätsdiagramm
API	Application Programming Interface
ASC	Automatische Stabilitäts Control
AWD	Anwendungsfalldiagramm
AWT	Abstract Window Toolkit
BA	Baustein Anforderung
BASt	Bundesanstalt für Straßenwesen
bbox	Bounding Box
BMBF	Bundesministerium für Bildung und Forschung
BMVBS	Bundesministerium für Verkehr, Bau und Stadtentwicklung
BMWi	Bundesministerium für Wirtschaft und Technologie
C2CC	Car2Car Communication
C2C-CC	Car2Car Communication Consortium

C2I	Car-to-Infrastructure
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CLR	Common Language Runtime
DEUFRAKO	Deutsch-Französische Kooperation
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DSC	Dynamic Stability Control
DSRC	Dedicated Short Range Communications
EA	Entwicklungszeit Anforderung
EAen	Entwicklungszeit Anforderungen
ESP	Elektronisches Stabilitätsprogramm
FCC	Federal Communication Commission
FCD	Floating Car Data
FUE	Fahrumgebungserfassung und Interpretation
FVM	Fahrerverhalten und Mensch-Maschine-Interaktion
GPL	GNU General Public License
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HDTraffic	High Definition Traffic
IEEE	Institute of Electrical and Electronics Engineers
IMP	Information Module Profile
IMP NG	IMP - Next Generation
INVENT	Intelligenter Verkehr und nutzergerechte Technik
ISO	International Organization for Standardization
IVHW	Inter-Vehicle Hazard Warning
J2ME	Java 2 Platform, Micro Edition

J2SE	Java 2 Platform, Standard Edition
Java EE	Java Platform, Enterprise Edition
Java ME	Java Platform, Micro Edition
Java SE	Java Platform, Standard Edition
JVM	Java Virtual Machine
KLD	Klassendiagramm
KOD	Komponentendiagramm
KSD	Kompositionsstrukturdiagramm
KVM	Kilobyte Virtual Machine
LCL	Location Code List
LGPL	GNU Library or Lesser General Public License
LK	Lane Keeping
MAC-Adresse	Media-Access-Control-Adresse
MIDP	Mobile Information Device Profile
NIV	Netzausgleich und Individualverkehr
NOW	Network on Wheels
OBU	On-board Unit
OSGi	Open Services Gateway initiative
OSM	OpenStreetMap
P2P-Systeme	Peer-to-Peer-Systeme
P2P-Systemen	Peer-to-Peer-Systemen
PAD	Paketdiagramm
PDA	Personal Digital Assistant
PKI	Public-Key-Infrastruktur
RM-ODP	Reference Model for Open Distributed Processing
RPC	Remote Procedure Call

PReVENT	Preventive and Active Safety Applications
RSU	Road-side Unit
SA	System Anforderung
SAX	Simple API for XML
SED	Sequenzdiagramm
SIM-TD	Sichere, intelligente Mobilität - Testfeld Deutschland
SLA	Service Level Agreement
SOA	Serviceorientierte Architekturen
SOAP	Simple Object Access Protocol
SODAD	Segment-Oriented Data Abstraction and Dissemination
SOTIS	Self-Organizing Traffic Information System
SP	SOTIS Process
SPs	SOTIS Processes
SSO	Single Sign-on
STA	Stauassistenz
TIC	Traffic Information Center
TMC	Traffic Message Channel
TTI	Travel and Traffic Information
UDDI	Universal Description, Discovery and Integration
UI	User Interface
UML	Unified Modeling Language
USDOT	United States Department of Transportation
VANET	Vehicular Ad Hoc Network
VAS	Vorausschauende Aktive Sicherheit
VED	Verteilungsdiagramm
VLA	Verkehrsleistungsassistenz

VM	Virtual Machine
VMTL	Verkehrsmanagement in Transport und Logistik
VRA	Verkehrliche Wirkung, Rechtsfragen und Akzeptanz
VSC	Vehicle Safety Communications
VSCC	VSC Consortium
WSDL	Web Service Description Language
XFCD	Extended Floating Car Data
XML	Extensible Markup Language
ZUD	Zustandsdiagramm

Literaturverzeichnis

[3SA02] 3SAT: *Internetseite: Floating Car Data gegen den Verkehrsinfarkt in Berlin*. http://www.3sat.de/nano/cstuecke/37788/index.html, 2002. – Abrufdatum: 20.05.2009

[AHS09] AHSRA: *Internetseite: Advanced Cruise-Assist Highway System Research Association*. http://www.ahsra.or.jp/index_e.html, 2009. – Abrufdatum: 23.05.2009

[AIS+77] ALEXANDER, Christopher ; ISHIKAWA, Sara ; SILVERSTEIN, Murray ; JACOBSON, Max ; FIKSDAHL-KING, Ingrid ; ANGEL, Shlomo: *A Pattern Language*. Oxford University Press, 1977

[And09a] ANDROID: *Internetseite: Android Developer*. http://developer.android.com/, 2009. – Abrufdatum: 17.09.09

[And09b] ANDROID: *Internetseite: Android Market: Erstmals über 10.000 Anwendungen verfügbar*. http://www.areamobile.de/news/11755-android-market-erstmals-ueber-10-000-anwendungen-verfuegbar, 2009. – Abrufdatum: 17.09.09

[And09c] ANDROID: *Internetseite: Location and Maps*. http://developer.android.com/guide/topics/location/index.html, 2009. – Abrufdatum: 17.09.09

[And09d] ANDROID: *Internetseite: The New York Times: Google: Expect 18 Android Phones by Year's End*. http://bits.blogs.nytimes.com/2009/05/27/google-expect-18-android-phones-by-years-end/, 2009. – Abrufdatum: 17.09.09

[And09e] ANDROID: *Internetseite: What is Android?* http://developer.android.com/guide/basics/what-is-android.html, 2009. – Abrufdatum: 17.09.09

[Arc09] ARCHITEKTURMUSTER: *Internetseite: Architekturmuster*. http://architekturmuster.de/index.php/Architekturmuster, Juni 2009. – Abrufdatum: 02.06.2009

[Bar02] BARLOCK, Pamela: *PDF: Kurzdarstellung des Teilprojektes Verkehrsleistungsassistenz VLA*. http://www.invent-online.de/downloads/VLA-handout-D.pdf, 2002. – Abrufdatum: 23.05.2009

[BAS02] BASt: *Internetseite: Bundesanstalt für Straßenwesen (BASt)*. http://www.bast.de/nn_42718/DE/Forschungsprojekte/abgeschlossene/fp-abgeschlossen-f4.html, 2002. – Abrufdatum: 23.05.2009

[BAS06] BASt: *Internetseite: Bundesanstalt für Straßenwesen (BASt) - Location Code List*. http://www.bast.de/cln_005/nn_42256/DE/Aufgaben/abteilung-v/referat-v2/Location-Code-List/location-code-list-start.html, 2006. – Abrufdatum: 23.05.2009

[BKPS04] BÖCKLE, Günter ; KNAUBER, Peter ; POHL, Klaus ; SCHMID, Klaus: *Software-Produktlinien - Methoden, Einführung und Praxis*. dpunkt.verlag, 2004

[BM04] BREDEMEYER, Dana ; MALAN, Ruth: *Software Architecture Action Guide*. 2004

[BM06] BREYMANN, Ulrich ; MOSEMANN, Heiko: *Java ME - Anwendungsentwicklung für Handys PDA und Co.* Hanser, 2006

[BMMM95] BROWN, William J. ; MALVEAU, Raphael C. ; MCCORMICK, Hays W. ; MOWBRAY, Thomas J.: *Anti Patterns - Refactoring Software, Architectures, and Projects in Crisis*. New York : John Wiley & Sons, 1995

[BMR+98] BUSCHMANN, Frank ; MEUNIER, Regine ; ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael: *Patter-orientierte Softwarearchitektur - Ein Pattern-System*. Addison-Wesley, 1998

[BMV08a] BMVBS: *Internetseite: Bundesministerium für Verkehr, Bau und Stadtentwicklung: TMC - Der Digitale Verkehrskanal*. http://www.bmvbs.de/dokumente/-,302.913019/Artikel/dokument.htm, September 2008. – Abrufdatum: 06.04.2009

[BMV08b] BMVBS: *Internetseite: Bundesregierung startet Forschungsprojekt SIM-TD zur Fahrzeugkommunikation*. http://www.bmvbs.de/-,302.1058854/doc.htm, 2008. – Abrufdatum: 23.06.2009

[BMW05] BMW: *Internetseite: Verkehrsinformationen mit allen Sinnen: BMW Group forscht an intelligenten Fahrzeugen als Verkehrsinformationssammler*. http://www.bmwgroup.com/d/0_0_www_bmwgroup_com/forschung_entwicklung/science_club/veroeffentlichte_artikel/2005/news200526.html, 2005. – Abrufdatum: 20.05.2009

[BMW06] BMW: *PDF: XFCD - Extended Floating Car Data: Potenziale und Durchdringungsraten*. http://www.bmwgroup.com/d/0_0_www_bmwgroup_com/forschung_entwicklung/mobilitaet_verkehr/verkehrsforschung/ExtendedFloatingCarData_d1.pdf, 2006. – Abrufdatum: 20.05.2009

[Bor09] BORCHERS, Detlef: *Internetseite: heise Autos News - Stauwarnung mittels Handydaten macht Fortschritte*. http://www.heise.de/autos/Stauwarnung-mittels-Handydaten-macht-Fortschritte--/artikel/s/7212, Januar 2009. – Abrufdatum: 23.05.2009

[Bos09] BOSCH: *Internetseite: Robert Bosch GmbH - Sensoren*. http://rb-k.bosch.de/de/leistungverbrauchemissionen/elektrik_steuerungen/sensoren/index.html, 2009. – Abrufdatum: 09.04.2009

[CAC09a] CACAO: *Internetseite: CACAO Licensing*. http://c1.complang.tuwien.ac.at/cacaowiki/CacaoLicense/, 2009. – Abrufdatum: 20.06.09

[CAC09b] CACAO: *Internetseite: Welcome to cacaovm.org!* http://www.cacaovm.org/, 2009. – Abrufdatum: 20.06.09

[Can07] CANALYS: *PDF: Canalys research release 2007/083*. http://www.canalys.com/pr/2007/r2007083.pdf, 2007. – Abrufdatum: 23.06.09

[CAS06] CASEGMBH: *PDF: C2BT2 - Bluetooth Adapter für den CAN-Bus (Technische Dokumentation)*. http://www.case-gmbh.de/DS4113D.pdf, August 2006. – Abrufdatum: 20.05.2009

[Des08] DESTATIS: *Internetseite: Statistisches Bundesamt Deutschland*. http://www.destatis.de, Oktober 2008. – Abrufdatum: 06.04.2009

[DEU03] DEUFRAKO: *PDF: Deutsch-Französische Kooperation auf dem Gebiet der Verkehrsforschung Laufende Projekte und Aktionen*. http://www.deufrako.org/pdf/flyer_a.pdf, 2003. – Abrufdatum: 23.05.2009

[DEU08] DEUFRAKO: *Internetseite: Deutsch-Französische Kooperation auf dem Gebiet der Verkehrsforschung Laufende Projekte und Aktionen*. http://deufrako.org/web/index.php?id=47&L=0, 2008. – Abrufdatum: 23.05.2009

[DK05] DANIEL, Christian ; KLEFFEL, Thomas: *PDF: Hacking TomTom GO*. http://events.ccc.de/congress/2005/fahrplan/attachments/569-Paper_HackingTomTomGo.pdf, 2005. – Abrufdatum: 23.06.09

[Drö03] DRÖSSER, Christoph: Ausgebremst - Formeln für den Stau. In: *DIE ZEIT* 26 (2003), Juni

[DT90] DORFMAN, Merlin ; THAYER, Richard H.: *Guidelines and Examples of System and Software Requirement Engineering*. Los Alamitos CA. : IEEE Computer Society Press, 1990

[Ecl09a] ECLIPSE: *Internetseite: Eclipse Pulsar*. http://www.eclipse.org/pulsar/, 2009. – Abrufdatum: 10.09.09

[Ecl09b] ECLIPSE: *Internetseite: Equinox Bundles*. http://www.eclipse.org/equinox/bundles/, 2009. – Abrufdatum: 23.06.09

[Ecl09c] ECLIPSE: *Internetseite: Swordfish SOA Runtime Framework Project*. http://www.eclipse.org/swordfish/, 2009. – Abrufdatum: 10.09.09

[FF+06] FREEMAN, Eric ; FREEMAN, Elisabeth u. a.: *Entwurfsmuster von Kopf bis Fuß*. O'Reily Verlag GmbH & Co. KG, 2006

[Fle02] FLEETNET: *PDF: Projektübersicht: FleetNet - Internet on the Road*. http://www.et2.tu-harburg.de/fleetnet/pdf/Fleetnet_deutsch.pdf, Mai 2002. – Abrufdatum: 23.05.2009

[Fri02] FRICKE, Volker: *Internetseite: Embedded Java und OSGi - Neue Technologien im Fahrzeug der Zukunft*. http://ls12-www.cs.tu-dortmund.de/teaching/courses/ss05/proseminars/automotive/download/literature/39_40a.pdf, 2002. – Abrufdatum: 23.06.09

[GHJV95] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralf ; VLISSIDES, John: *Design Patterns*. Addison-Wesley, 1995

[GNU09a] GNU: *Internetseite: GNU Classpath*. http://www.gnu.org/software/classpath/, 2009. – Abrufdatum: 20.06.09

[GNU09b] GNU: *Internetseite: GNU Classpath*. http://www.gnu.org/software/classpath/license.html, 2009. – Abrufdatum: 20.06.09

[Gra97] GRADY, Booch: *Objektorientierte Analyse und Design*. 4., unveränderter Nachdruck. Addison-Wesley, 1997

[Gut84] GUTTMAN, A.: R-Trees: a dynamic index structure for spatial searching / University of California Berkely. Boston, USA, 1984. – Forschungsbericht. – SIGMOD Conference

[Ham05] HAMMERSCHALL, Ulrike: *Verteilte Systeme und Anwendungen*. Pearson Studium, 2005

[Har01] HARKER, B. J.: PROMOTE-Chauffeur II - vehicle to vehicle communications systems. In: *ADAS01: International Conference on Advanced Driver Assistance Systems* Central Research Laboratories Limited, 2001, S. 81–85

[HL07] HACK, Stefan ; LINDEMANN, Markus A.: *Enterprise SOA einführen*. Bonn : Galileo Press, 2007

[HMGRZ06] HERDEN, Sebastian ; MARX GÓMEZ, Jorge ; RAUTENSTRAUCH, Claus ; ZWANZIGER, André: *Software-Architekturen für das E-Business - Enterprise-Application-Integration mit verteilten Systemen*. Springer Berlin Heidelberg, 2006

[Hof09] HOFMANN, Moritz: *Infrastruktur für die kooperative Situationsanalyse in dezentralen Verkehrsleitsystemen (SOTIS)*, Technische Universität Dortmund - Fachbereich Informatik, Diplomarbeit, 2009

[HTKR05] HÖPFNER, Hagen ; TÜRKER, Can ; KÖNIG-RIES, Birgitta: *Mobile Datenbanken und Informationssysteme - Konzepte und Techniken*. dpunkt.verlag, 2005

[IEE00] IEEE: *Recommended Practice for Architectural Description of Software-Intensive Systems*. 2000

[ISO98] ISO10746: *Internetseite: International Organization for Standardization - Information technology - Open Distributed Processing-Reference model: Overview*. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=20696, 1998. – Abrufdatum: 02.06.2009

[IST00] ISTWORLD: *Internetseite: Promote Chauffeur II - Project Status*. http://www.ist-world.org/, 2000. – Abrufdatum: 23.05.2009

[JSI09] JSI: *Internetseite: Java Spatial Index (RTree) Library*. http://sourceforge.net/projects/jsi/, 2009. – Abrufdatum: 22.07.09

[KH03] KROLL, Michael ; HAUSTEIN, Stefan: *J2ME Developers's Guide - Java Anwendungen für mobile Geräte*. Markt+Technik, 2003

[KIT09] KIT: *Internetseite: Performance-Messung*. http://sdqweb.ipd.uka.de/wiki/Performance-Messung, 2009. – Abrufdatum: 10.09.09

[Kle82] KLEBELSBERG, Dieter: *Verkehrspsychologie*. Springer-Verlag, 1982

[Küh07] KÜHNEL, Andreas: *Visual Basic 2005 - Das umfassende Handbuch*. Galileo Computing, 2007

[Lin09] LINUX: *Internetseite: Linux is Everywhere: An Overview of the Linux Operating System*. http://www.linux.com/learn/resource-center/376-linux-is-everywhere-an-overview-of-the-linux-operating-system, 2009. – Abrufdatum: 23.06.09

[LW09] LINUX-WIKI: *Internetseite: /proc/<PID>*. http://linuxwiki.de/proc/pid, 2009. – Abrufdatum: 10.09.09

[M+00] MORRIS, R. u. a.: A scalable ad hoc wireless network system. In: *Proc. 9th ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System*, Kolding, September 2000

[M+07] MELZER, Ingo u. a.: *Service-orientierte Architekturen mit Web Services*. 2. Auflage. München : Elservier GmbH, Spektrum Akademischer Verlag, 2007

[Mae09] MAEMO: *Internetseite: Maemo*. http://maemo.org/, 2009. – Abrufdatum: 20.06.09

[Mag04] MAGIERA, Przemyslaw: *Reputationssysteme in großen, hoch-mobilen Ad-hoc-Netzen*, Technische Universität Darmstadt - Fachbereich Informatik, Diplomarbeit, 2004

[Mic09a] MICROSOFT: *Internetseite: Externe Ressourcen für .NET Compact Framework*. http://msdn.microsoft.com/de-de/library/ms172552(VS.85).aspx, 2009. – Abrufdatum: 23.06.09

[Mic09b] MICROSOFT: *Internetseite: .NET Compact Framework*. http://msdn.microsoft.com/de-de/library/f44bbwa1(VS.85).aspx, 2009. – Abrufdatum: 23.06.09

[Mic09c] MICROSOFT: *Internetseite: .NET Compact Framework-Architektur*. http://msdn.microsoft.com/de-de/library/9s7k7ce5(VS.85).aspx, 2009. – Abrufdatum: 23.06.09

[Mic09d] MICROSOFT: *Internetseite: Visual Studio und .NET Compact Framework*. http://msdn.microsoft.com/de-de/library/ms172551(VS.85).aspxx, 2009. – Abrufdatum: 23.06.09

[Mil56] MILLER, George A.: *2*. Bd. 63: *The Magical Number Seven, Plus Or Minus Two: Some Limits on Our Capacity for Proceesing Information*. The Psychological Review, 1956

[MM01] MALVEAU, Raphael ; MOWBRAY, Thomas: *Software Architect Bootcamp*. London : Prentice Hall, 2001

[MP09] MONO-PROJEKT: *Internetseite: Willkommen auf mono-project.de - der deutschen Mono Community*. http://www.mono-project.de/, 2009. – Abrufdatum: 23.06.09

[MPW07] MCLAUGHLIN, Brett D. ; POLLICE, Gary ; WEST, David: *Objektorientierte Analyse und Design von Kopf bis Fuß*. O'Reily Verlag GmbH & Co. KG, 2007

[MWH01] MAUVE, M. ; WIDMER, J. ; HARTENSTEIN, H.: A survey on position-based routing in mobile ad hoc networks. In: *IEEE Networks* Bd. 16, nr. 6, 2001

[N8108a] N810: *Internetseite: How to become root on a N810 with the latest OS2008 update*. http://tabletoid.blogspot.com/2008/01/how-to-become-root-on-n810-with-latest.html, 2008. – Abrufdatum: 20.06.09

[N8108b] N810: *Internetseite: N810 - First steps for Java developers*. http://frapaa.blogspot.com/2008/05/n810-first-steps-for-java-developers.html, 2008. – Abrufdatum: 20.06.09

[Nag95] NAGEL, Kai: *High-speed microsimulations of traffic flow*, Universität zu Köln, Inaugural-Dissertation, 1995

[NAV08] NAVTEQ: *Internetseite: Verkehrsinformationen - Freie Fahrt mit TMCpro und HD-Traffic*. http://www.tmcpro-info.de/files/200802/20080815_PM_TMCpro_2_Features.pdf, Februar 2008. – Abrufdatum: 23.05.2009

[Nok09a] NOKIA: *Internetseite: Nokia N810*. http://www.nokia.de/produkte/mobiltelefone/nokia-n810/funktionen, 2009. – Abrufdatum: 20.06.09

[Nok09b] NOKIA: *Internetseite: Nokia N95*. http://www.nokia.de/produkte/mobiltelefone/nokia-n95, 2009. – Abrufdatum: 20.05.09

[NS92] NAGEL, Kai ; SCHRECKENBERG, Michael: *A cellular automaton model for freeway traffic*. September 1992

[Ope09] OPENTOM: *Internetseite: Main Page*. http://www.opentom.org/, 2009. – Abrufdatum: 23.06.09

[OSG07a] OSGI: *PDF: Technical Whitepaper - About the OSGi Service Platform*. http://www.osgi.org/wiki/uploads/Links/OSGiTechnicalWhitePaper.pdf, 2007. – Abrufdatum: 23.06.09

[OSG07b] OSGI, Alliance: *PDF: OSGi Service Platform Release 4 Version 4.0 Compendium Specification*. http://www.osgi.org/Download/File?url=/download/r4v40/r4.cmpn.pdf, 2007. – Abrufdatum: 23.06.09

[OSG09a] OSGI: *Internetseite*. http://www.osgi.org/, 2009. – Abrufdatum: 23.06.09

[OSG09b] OSGI: *Internetseite: About Technology*. http://www.osgi.org/About/Technology, Juni 2009. – Abrufdatum: 23.06.09

[OSG09c] OSGi: *Internetseite: Automotive Electronics Market*. http://www.osgi.org/Markets/Automotive, Juni 2009. – Abrufdatum: 23.06.09

[OSM09a] OSM: *Internetseite: FAQs: Fragen und Antworten*. http://www.openstreetmap.de/faq.html, 2009. – Abrufdatum: 22.07.09

[OSM09b] OSM: *Internetseite: OpenStreepMap*. http://www.openstreetmap.org/, 2009. – Abrufdatum: 22.07.09

[OSM09c] OSM: *Internetseite: OSM Protocol Version 0.6*. http://wiki.openstreetmap.org/wiki/OSM_Protocol_Version_0.6, 2009. – Abrufdatum: 22.07.09

[OSM09d] OSM: *Internetseite: OSM Tipps*. http://osmtipps.lefty1963.de/, 2009. – Abrufdatum: 22.07.09

[OSM09e] OSM: *Internetseite: QuadTiles*. http://wiki.openstreetmap.org/wiki/QuadTiles, 2009. – Abrufdatum: 22.07.09

[OSM09f] OSM: *Internetseite: Wiki Dortmund*. http://wiki.openstreetmap.org/wiki/Dortmund, 2009. – Abrufdatum: 22.07.09

[Par71] PARNAS, David L.: Information distribution aspects of design methodology / Depart. Computer Science, Carnegie-Mellon U. Pittsburgh, Pa., 1971. – Tech. Rept.

[Par72] PARNAS, David L.: *On the Criteria To Be Used in Decomposing Systems into Modules*. Association for Computing Machinery, 1972

[Plu05] PLUM: *PDF: NOW: Network on Wheels - Übersicht*. http://www.network-on-wheels.de/downloads/overview_de.pdf, 2005. – Abrufdatum: 23.05.2009

[PW92] PERRY, Dewayne E. ; WOLF, Alexander L.: *Foundations for the Study of Software Architectur*. ACM SIGSOFT Software Engineering Notes, 1992

[Rad08] RADERMACHER, W.: *Statistisches Jahrbuch 2008*. Statistisches Bundesamt, Wiesbaden, 2008. – 439–440 S.

[Rec91] RECHTIN, Eberhard: *Systems Architecting - Creating and building complex systems*. CRC Press, 1991

[RH09] REUSSNER, Ralf ; HASSELBRING, Wilhelm: *Handbuch der Software-Architektur*. dpunkt.verlag, 2009. – 2., überarbeitete und erweiterte Auflage

[RMM+02] REICHARDT, D. ; MIGLIETTA, M. ; MORETTI, L. ; P., Morsink ; SCHULZ, W.: CarTALK 2000: safe and comfortable driving based upon inter-vehicle-communication. In: IEEE (Hrsg.): *Intelligent Vehicle Symposium* Bd. 2. DaimlerChrysler AG, Germany, Juni 2002, S. 545–550

[RR06] ROBERTSON, Suzanne ; ROBERTSON, James: *Mastering the Requirements Process*. Harlow : Addison Wesley, 2006

[Sch07] SCHMATZ, Klaus-Dieter: *Java Micro Edition - Entwicklung mobiler JavaME-Anwendungen mit CLDC und MIDP*. Bd. 2., aktualisierte und erweiterte Auflage. dpunkt.verlag, 2007

[SD07] SHULMAN, Michael ; DEERING, Richard: *PDF: Vehicle safety communications in the United States*. http://www-nrd.nhtsa.dot.gov/pdf/nrd-01/esv/esv20/07-0010-O.pdf, 2007. – Abrufdatum: 23.05.2009

[SMI+08] SCHULZE, Matthias ; MÄKINEN, Tapani ; IRION, Joachim ; FLAMENT, Maxime ; KESSEL, Tanja: *PDF: IP PReVENT Final Report*. http://www.prevent-ip.org/download/deliverables/IP_Level/PR-04000-IPD-080222-v15_PReVENT_Final_Report_Amendments%206%20May%202008.pdf, Mai 2008. – Abrufdatum: 23.05.2009

[Son09] SONYERICSSON: *Internetseite: Xperia X1*. http://www.sonyericsson.com/x1, 2009. – Abrufdatum: 20.05.2009

[Spi99] SPIEGEL: *Internetseite: Täglich 1000 Kilometer Stau in Deutschland*. http://www.spiegel.de/auto/aktuell/0,1518,42281,00.html, September 1999. – Abrufdatum: 06.04.2009

[Spi06] SPIEGEL: *Internetseite: Stillstand auf der Straße - Von Deutschland bis Australien im Stau*. http://www.spiegel.de/auto/aktuell/0,1518,440367,00.html, Oktober 2006. – Abrufdatum: 06.04.2009

[SUN05] SUN: *Internetseite: Sun Microsystems*. http://java.sun.com/developer/technicalArticles/JavaOne2005/naming.html, 2005. – Abrufdatum: 23.06.2009

[SUN06] SUN: *Internetseite: Sun Microsystems*. http://java.sun.com/javase/namechange.html, 2006. – Abrufdatum: 23.06.2009

[SUN07] SUN: *PDF: Sun Microsystems*. http://de.sun.com/sunnews/events/2007/20071203/pdf/TD_FRA_GoslingKeynote.pdf, 2007. – Abrufdatum: 23.06.2009

[Tom09] TOMTOM: *Internetseite: source code*. http://www.tomtom.com/page.php?Page=gpl, 2009. – Abrufdatum: 23.06.09

[Tur00] TURKSMA, S.: The various uses of floating car data. In: *Road Transport Information and Control*, 2000, S. 51–55. – Tenth International Conference on (Conf. Publ. No. 472)

[Ull09] ULLENBOOM, Christian: *Java ist auch eine Insel - Programmieren mit der Java Standard Edition Version 6*. Galileo Computing, 2009

[VAC+05] VOGEL, Oliver ; ARNOLD, Ingo ; CHUGHTAI, Arif ; IHLER, Edmund ; MEHLIG, Uwe ; NEUMANN, Thomas ; VÖLTER, Markus ; ZDUN, Uwe: *Software-Architektur Grundlagen-Konzepte-Praxis*. Elservier GmbH, Spektrum Akademischer Verlag, 2005

[Ver07] VERIVOX: *Internetseite: Navigationsanbieter will Handy-Daten für Verkehrsservice nutzen*. http://www.verivox.de/mobile/article.aspx?i=21283, November 2007. – Abrufdatum: 23.05.2009

[VKZ04] VÖLTER, Markus ; KIRCHNER, Michael ; ZDUN, Uwe: *Remoting Patterns - Foundations of Enterprise, Internet, and Realtime Distributed Object Middleware*. John Wiley & Sons, 2004

[Web09] WEBSTER: *Internetseite: Webster's New World College Dictionary - architecture definition*. http://www.yourdictionary.com/architecture, 2009. – Abrufdatum: 02.06.2009

[WER+03] WISCHHOF, Lars ; EBNER, Andre ; ROHLING, Hermann ; LOTT, Matthias ; HALFMANN, Rüdiger: *SOTIS - A Self-Organizing Traffic Information System*. 2003

[WER04] WISCHHOF, Lars ; EBNER, Andre ; ROHLING, Hermann: Self-Organizing Traffic Information System based on Car-to-Car Communication: Prototype Implementation. In: *1st International Workshop on Intelligent Transportation, Hamburg* (2004)

[WER+05] *Kapitel* Self-Organizing Traffic Information System. In: WISCHHOF, Lars ; EBNER, Andre ; ROHLING, Hermann ; LOTT, Matthias ; HALFMANN, Rüdiger: *Inter-Vehicle-Communications Based on Ad Hoc Networking Principles*. Universitätsverlag Karlsruhe, 2005, S. 233–258

[WHKL08] WÜTHERICH, Gerd ; HARTMANN, Nils ; KOLB, Bernd ; LÜBKEN, Matthias: *Die OSGI Service Platform - Eine Einführung mit Eclipse Equinox*. dpunkt.verlag, 2008

[Wie03] WIEGERS, Karl E.: *Software Requirements*. Second Edition. Microsoft Press, 2003

[Wik09] WIKIPEDIA: *Internetseite: Design pattern (computer science)*. http://en.wikipedia.org/wiki/Design_pattern_(computer_science), Juni 2009. – Abrufdatum: 23.06.2009

[Wis07] WISCHHOF, Lars: *Self-Organizing Communication in Vehicular Ad Hoc Networks*, Technische Universität Hamburg-Harburg, Diss., 2007

[WT05] WIETZKE, Joachim ; TIEN, Tran M.: *Automotive Embedded Systeme*. Berlin, Heidelberg : Springer, 2005 (Xpert.press)

[YC87] YOURDON, E. ; CONSTANTINE, L.: *Structured Design: Fundamentals of a Discipline of Computer Programming and Design*. Prentice Hall, 1987

[Z+07] ZABLER, Erich u. a.: *Sensoren im Kraftfahrzeug*. Robert Bosch GmbH, 2007

[Zac87] ZACHMAN, John A.: *A Framework for Information System Architecture*. IBM Publication, 1987