Please quote as: Hoffmann, H.; Leimeister, J. M. & Krcmar, H. (2008): Anforderungen an Werkzeuge zum Prototyping von Automotive Services. In: Multikonferenz Wirtschaftsinformatik (MKWI) 2008, München, Germany.

Anforderungen an Werkzeuge zum Prototyping von Automotive Services

Holger Hoffmann, Jan Marco Leimeister, Helmut Krcmar

Lehrstuhl für Wirtschaftsinformatik
Technische Universität München
Boltzmannstr. 3
D-85748 Garching b. München
holger.hoffmann@in.tum.de
leimeister@in.tum.de
krcmar@in.tum.de

Abstract: Bei der Konzeption neuer Services (z.B. dynamische Navigationssysteme) in Fahrzeugen stehen die Entwickler meist vor ähnlichen Problemstellungen, wie z.B. der komplexen Infrastruktur im Fahrzeug oder der nötigen Reduktion der Fahrerablenkung bei der Nutzung während der Fahrt. Für die prototypische Umsetzung von innovativen mobilen Diensten, z.B. im Rahmen von Pilotierungsprojekten, kann ein „Komponentenbaukasten", der Entwicklern vorgefertigte Module liefert, hier Abhilfe schaffen. Die Entwickler sparen so wertvolle Entwicklungszeit, da häufig verwendete Bausteine automobiler Dienste (z.B. Mensch-Maschine Schnittstellen) „off the shelve" benutzt werden können. Wir zeigen die Anforderungen an eine solche Prototyping Plattform auf und beschreiben, wie die Architektur gestaltet sein könnte und welche Komponenten Teil einer solchen Umgebung sein könnten.

1 Einführung

Software ist heute bereits ein wesentlicher Bestandteil im Wertschöpfungsanteil von Oberklassefahrzeugen neuerer Generation [MeFF04]. Das Volumen an Softwarekomponenten, meist für die Benutzung auf embedded devices entwickelt, steigt dabei stetig an. Während das BMW 7er Modell in 2003 noch rund 60 MB Software enthielt [Saad03], wird für die kommende Generation von Audi A6 bereits das Vierfache an Software erwartet.

Für diesen Anstieg gibt es zweierlei Gründe. Zum einen werden immer mehr Funktionen die bislang elektronisch bereitgestellt wurden nun als Softwarefunktionen umgesetzt, so wie früher elektronische Lösungen die mechanischen Funktionen ersetzt haben [Saad03]. So mussten früher die Autofenster manuell „gekurbelt" werden, konnten dann über einen Elektromotor verstellt werden und sind jetzt zentral ansteuerbar, so dass sie z.B. beim Einschalten der Umluftfunktion automatisch geschlossen werden können. Zum anderen steigt der Grad der Vernetzung zwischen den einzelnen Funktionen im Auto immer weiter an, was zu einem komplexen Gesamtsystem führt [Saad03]. Zusammen mit der Variantenvielfalt wählbarer Fahrzeuge, und damit verbauter Soft- und Hardware, und

deren asynchronen Lebenszyklen ergibt sich ein sehr vielfältiges Zielsystem. Daraus resultiert das Problem, dass Entwickler von automobilen Anwendungen nie vom Vorhandensein bestimmter Konfigurationen bei der Umsetzung Ihrer Anwendungen ausgehen können.

Zusätzlich gab es in der Vergangenheit weitere Probleme bei der Einführung softwarebasierter Dienste im Automobil in Deutschland. So haben z.B. fast alle Automobilhersteller ihr Angebot an Telematikdienstleistungen eingestellt. Als Hauptgründe für deren Scheitern am Markt werden meist die folgenden Gründe genannt: die Kommunikationskosten waren zu hoch [Fros03], die angebotenen Dienste trafen nicht wirklich die Bedürfnisse der Kunden [Fuhr01];[Werd05], Telematikdienste waren vor allem in der Entwicklung zu sehr auf Technologien fixiert [Werd05], die mangelhafte Berücksichtigung ökonomischer Aspekte machte es fast unmöglich langfristig erfolgreiche Dienste anzubieten.

Als ein möglicher Lösungsansatz für eine möglichst Erfolg versprechende systematische Entwicklung innovativer mobiler Dienste für den Automobilsektor, wurde das MACS Design Framework und ein dazu passendes Vorgehensmodell entwickelt. Die Erfahrungen aus einer ersten Erprobung des Design Frameworks legen den Schluss nahe, dass die für den Entwicklungsprozess vorgestellten Ideen für eine Prototyping-Plattform grundsätzlich als Abstraktionsschicht im Automobil geeignet sind [HoLK07b]. Um dieses Framework und den dazugehörigen Entwicklungsprozess durch geeignete Werkzeuge zu unterstützten stellt sich zunächst Fragen nach den Anforderungen an eine Entwicklungsumgebung für Prototypen im Automobil und wie diese gestaltet sein muss um Dienste schon in frühen Entwicklungsstadien im Fahrzeug erproben bzw. „erfahrbar“ machen zu können.

Im folgenden Beitrag möchten wir nach einer kurzen Zusammenfassung relevanter Vorarbeiten in Form des MACS Design Framework und des Vorgehensmodells, beschreiben um darauf aufbauend einen Lösungsvorschlag für eine entsprechend abgestimmte Werkzeugunterstützung in einem Teilbereich des Design Frameworks, nämlich das Prototyping, vorzustellen. Hierzu stellen wir die Anforderungen an ein solches Werkzeug dar und stellen als Abschluss erste Lösungsansätze für eine Architektur vor, die diesen Anforderungen genügen kann.

2 Methodische Entwicklung von Diensten für das Automobil

Im folgenden Abschnitt stellen wir kurz einen Ansatz zur methodischen Entwicklung von Diensten im Automobil vor (vgl. [HoLK07b]).

Das MACS Design Framework in folgende Teilbereiche (bzw. Schichten) untergliedert, die jeweils einen Schritt im Entstehungszyklus eines mobilen Dienstes beschreiben:

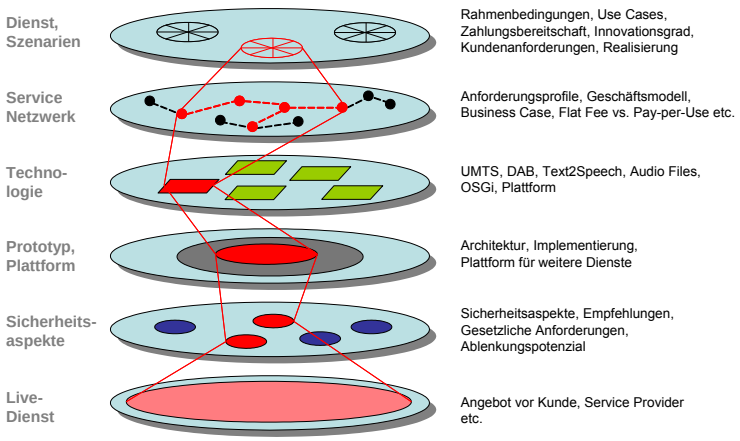

Abbildung 1: MACS Design Framework für Automotive Services [HoLK07]

Diensteszenarien umfassen den allgemeinen Aufbau und Rahmen, in dem der Dienst zum Einsatz kommen soll, sowie Use Cases für den mobilen Dienst. Eine Analyse der Diensteszenarien nach benötigten Partnern zur Diensteerbringung ergibt Informationen in welcher Form die einzelnen Partner im *Wertschöpfungsnetzwerk* interagieren können und wie das Wertschöpfungsnetz aufgebaut sein kann bzw. soll.

Eine der größten Herausforderungen für die Auswahl passender *Technologien* im Automobilbereich sind die unterschiedlich langen Lebenszyklen des Automobils und der im Fahrzeug verwendeten Software [Hart04]. Um eine tragfähige Auswahl an Technologien und die Erstellung einer Infrastruktur zur Kompensierung des „Lifecycle Mismatch“ [Moha06] zwischen Software und Automobil zu ermöglichen, ist besonderes Augenmerk auf den Bereich des *Prototyping* zu legen. Hinzu kommt die notwendige Betrachtung der *Sicherheitsaspekte* zur Gewährleistung der sicheren Benutzbarkeit des Dienstes während der Fahrt.

Ist der Dienst für straßentauglich befunden, folgt die Planung des *Roll-Out des Live Dienstes*. Dazu dienen die technische Produktbeschreibung und ein positiver Business-Case als Basis für den herstellerspezifischen Produktentwicklungsprozess (PEP).

3 Anforderungen an das Software-Prototyping im Automobil

Um den Teilbereich des Prototypings im vorangegangenen Abschnitt präsentierten Design Frameworks unterstützen zu können ist geplant, eine Rapid-Prototyping Plattform aufzubauen.

Die Vision dieser Plattform ist es, den Dienste-Entwicklern eine *einheitliche API* zur Verfügung zu stellen, mit der sich *interaktive* mobile Services im *Auto* prototypisch umsetzen lassen, um z.B. in Pilotierungsprojekten eingesetzt zu werden.

Im folgenden Abschnitt stellen wir kurz die für ein solches Werkzeug gefundenen Anforderungen vor.

3.1 Anforderungen und Gestaltungsziele einer Rapid Prototyping Plattform für Automotive Services auf Literaturbasis

Aus der Vision für die Plattform lassen sich folgende (nicht komplett disjunkte) Bereiche extrahieren, aus denen funktionale und nicht funktionale Anforderungen und Gestaltungsziele an die zu erstellende Lösung erhoben werden.

- *Prototyping Plattform als API*: Was sind good practices (Erfahrungswerte, Erfolgsfaktoren, etc.) bei der Konzeption einer API?
- *Prototyping Plattform als Werkzeug zur Erstellung interaktiver Dienste*: Wie muss ein Erstellungswerkzeug für interaktive Software gestaltet sein?
- *Prototyping Plattform als Basis für Software im Automobil*: Welche besonderen Ansprüche werden an Software im Automobil gestellt?

Anforderungen an die Prototyping Plattform als API

Eine gute API zeichnet aus, dass man kaum etwas von ihr bemerkt wenn man sie benutzt. Sie bietet zur richtigen Zeit für eine definierte Aufgabe eine definierte Schnittstelle, der Aufruf dieser Schnittstelle gestaltet sich einfach, d.h. die Benutzung ist intuitiv und gut dokumentiert. Darüber hinaus stellt die API sicher, dass Randbedingungen (und Invarianten) eingehalten werden [Henn07, S. 25]. Eine schlecht entworfene API ist oft schwerer zu verstehen und/oder weist funktionale Schwächen auf [Scaf06]. Dadurch können diverse Probleme in der Umsetzung von Anwendungen entstehen und zu langwierigen Problemen führen [Bloc06]. So muss oft zusätzlicher Programmcode geschrieben werden, um funktionale Schwächen auszugleichen [Henn07, S. 29].

Eine der zentralen Anforderungen ist, dass eine API dem Benutzer ausreichend Funktionalität bereitstellen muss, um sein Ziel zu erreichen [Henn07]. Dabei soll die API soll so klein wie möglich gehalten werden [Henn07]. Im Zweifel soll Funktionalität nicht in der API implementiert werden [Bloc06].

APIs sollten aus Perspektive der zukünftigen Nutzer entworfen werden [Henn07]. Gute APIs „geben den schwarzen Peter nicht weiter“ [Henn07], d.h. es werden z.B. Verfahrenweisen für die Benutzung festgelegt. Der Entwurf erfolgt bewusst, und nicht als „design by accident“. Gute APIs beachten die Benutzbarkeit [Henn07], d.h. auch große APIs sind in der Benutzung konsistent.

APIs können nicht ohne Verständnis ihres Kontextes entworfen werden [Henn07]. Universell einsetzbare APIs sollen keine Verfahrensweise als Grundlage annehmen, spezialisierte APIs hingegen sollen dies tun [Henn07]. Als Grundlage für das Verständnis des Kontextes werden vor der Umsetzung der API zunächst Anforderungen

von späteren Anwendern erhoben [Bloc06]. Das API Design selbst soll während der Entwicklung möglichst breit diskutiert wird.

Ein weiterer wichtiger Punkt ist die Dokumentation der API für deren Benutzer. Dabei ist wichtig, dass alle Bestandteile der API dokumentiert werden [Bloc06], möglichst bevor sie implementiert werden [Henn07]. Eine Möglichkeit der Dokumentation ist die Bereitstellung von kleinen Programmbeispielen, aus denen die Benutzung ersichtlich wird [Scaf06]. Diese Programmbeispiele müssen mustergültig umgesetzt werden, da sie vielen Anwendern als Grundgerüst dienen werden [Bloc06].

Prototyping Plattform als Werkzeug zur Gestaltung interaktiver Software

Die Anforderungen an die Plattform als Werkzeug zur Gestaltung interaktiver Software lassen sich aus zwei Richtungen ermitteln. Einerseits gibt es die Möglichkeit, Anforderungen an derartige Werkzeuge zu ermitteln. Andererseits ist es möglich, aus den Anforderungen an die zu erstellenden Produkte abzuleiten, wie ein Werkzeug gestaltet sein muss, um zu ermöglichen, dass mit ihm erstellte Produkte den Anforderungen genügen können.

Direkte Anforderderungen an ein Werkzeug zur Oberflächengestaltung haben Desmarais et al. [DHJK94] in einer Studie mit 56 Endnutzern erhoben (s. Tabelle 1).

Merkmal \ Wichtigkeit	**sehr**	**ziehmlich**	**unwichtig**
Performance (fehlerfrei und zuverlässig)	43	10	0
Einsetzbar für das Rapid Prototyping	38	13	2
Einsetzbar für Entwicklung von Endprodukten	36	13	4
Inhalt & Qualität der Dokumentation	32	17	4
Modulare Entwicklung	32	20	0
Einfachheit der Benutzung	31	22	0
Wiederverwendbarkeit der Ergebnisse	30	20	1
Ausführungsgeschwindigkeit	29	24	0
Einfache Verknüpfung mit anderen Werkzeugen	26	21	5
Verfügbarkeit von integrierbaren Bibliotheken	25	22	5
Fehlererkennung / -beseitigung	22	27	4
Technischer Support verfügbar	21	30	2
Durchsetzen von Schnittstellen-Standards	21	22	6
On-line Hilfe im Werkzeug	20	21	12
Einfach zu erlernen und zu installieren	19	29	5
Portabilität des Werkzeuges	15	27	11
Anpassbarkeit an eigene Bedürfnisse	11	28	14
Code Erstellung für versch. Programmiersprachen	11	15	26
Kosten des Werkzeuges	8	38	6
Im Projektmanagement verwendbar	6	19	22

Tabelle 1: Rangreihung der Wichtigkeit einiger Merkmale von Werkzeugen zur Oberflächengestaltung. [DHJK94, S. 280]

Bemerkenswert ist, dass mit den Einsatzbereichen für das Prototyping und für die Produktion zwei gegensätzliche Ziele als sehr wichtig bewertet wurden. Insgesamt werden allgemein übliche Qualitätsansprüche an das Werkzeug gestellt, so z.B. dass das Werkzeug schnell, fehlerfrei und zuverlässig arbeitet, sowie über eine gute Dokumentation verfügt und einfach zu bedienen ist. Einen wichtigen Hinweis auf die Strategien der Projektteams in Bezug auf die Komplexität liefern die Punkte der „modularen Entwicklung“ und der Wiederverwendbarkeit der Ergebnisse.

Indirekte Anforderungen an die Plattform, d.h. Anforderungen, die sich aus den zu erstellenden Diensten ergeben, können aus den Anforderungen an Software ermittelt werden wie sie in den Bereichen des User Interface Design (z.B. [HiHa93; ShRP07; ZCTT05]) und in Bereichen der Ergonomie (z.B. [Schm93]) aufgestellt werden.

Allgemeine Anforderungen an Software aus Sicht des User Interface Design lassen sich anhand der Definition von „Usability“ [Niel93, S. 26] erkennen. Die Benutzung eines Systems sollte *leicht zu erlernen* sein, um eine hohe Produktivität zu ermöglichen soll ein System *effizient* zu nutzen sein, die Funktionen eines Systems sollen, z.B. für Gelegenheitsnutzer, *leicht zu merken* sein. Darüber hinaus soll Software eine *geringe Fehlerrate* haben und Fehler leicht wieder zu beheben sein. Mit diesen Punkten soll insgesamt eine hohe *subjektive Zufriedenheit* der Benutzer mit dem System erreicht werden. Diese Anforderungen werden an vielen Stellen in Bereichen der Ergonomie aufgegriffen, konkretisiert und teilweise um Lösungsvorschläge erweitert [Bubb93; Krais93; Rühm93].

Eine zuverlässige und stichhaltige Sicherheitsevaluation[1] für neu entwickelte mobile Dienste ist unerlässlich [ShRP07; WiHe07]. Diese ist jedoch nur in der vorgesehenen Umgebung, d.h. im Automobil, sinnvoll. Das bedeutet, dass spätestens für die Sicherheitsevaluation ein im Fahrzeug testbarer Prototyp verfügbar sein muss [Nied03].

Prototyping Plattform als Werkzeug zur Erstellung von “Automotive Software”

Software ist ein wesentlicher Bestandteil im Wertschöpfungsanteil von Oberklassefahrzeugen neuerer Generation [MeFF04]. Das Volumen an Softwarekomponenten, meist für die Benutzung auf embedded devices entwickelt, steigt bedingt durch die Umsetzung von vormals elektronischen Funktionen in Softwarefunktionen stetig an [Saad03]. Während das BMW 7er Modell in 2003 noch rund 60 MB Software enthielt [Saad03], wird für die kommende Generation von Audi A6 bereits das Vierfache an Software erwartet und für 2010 eine Gesamtgröße von einem Gigabyte vorhergesagt [PBKS07].

Die Anforderungen an die Konstruktion von Software für das Automobil sind geprägt von den im Zielsystem zu findenden Extremen. Wesentlichen Einfluss nehmen unter anderem folgende Eigenschaften [Broy03; EAGS05; Saad03]:

[1] Sicher im Sinn von „gefahrlos in der Benutzung“

- Sicherheitskritische Funktionen (Safety)
- Unterschiedlich lange Lebenszeiten & Lebenszyklen mit unzuverlässiger Wartung
- Variantenvielfalt der Systeme & Komplexität des Zielsystemes
- Unterschiedliche Benutzergruppen (z.B. Fahrer, Beifahrer; verschiedene Zielmärkte)

Die Sicherheit des Benutzers von Software im Fahrzeug ist einer der wesentlichen Unterscheidungspunkte zwischen automotive Software und Standardsoftware auf einem anderen PC. Neben allgemeinen Anforderungen zum Thema „Usability“ ergeben sich für interaktive Dienste im Automobil zusätzliche Anforderungen aus der Nutzungssituation heraus, welche die sichere Benutzbarkeit der Anwendung garantieren sollen [HiSS07]. So müssen beispielsweise gesonderte rechtliche Regelungen eingehalten werden, wenn Software während des Fahrbetriebes genutzt wird. Da innovative Dienste oftmals in einen Bereich fallen, der in dieser Form noch nicht vom Gesetzgeber abgedeckt wird, ist eine konkrete Handlungsrichtlinie allerdings oft schwer zu definieren.

Neben den Sicherheitsbetrachtungen hat der „lifecycle mismatch“ zwischen Automobil und Service mit den größten Einfluss auf die Gestaltung innovativer Services [Hart04; HiSS07; Moha06]. Es existieren über die verschiedenen Modelle der einzelnen Hersteller verschiedene Architekturen für Software im Fahrzeug, bestehend aus Hardware (z.B. embedded devices, Bus-Systeme) und Software (z.B. Betriebssystem für zentrale Rechner), die jeweils eigenen Lebenszyklen unterliegen. So sind häufig verschiedene Revisionen der zugrunde liegenden Hardwareplattformen, Betriebssysteme und Softwarearchitekturen zu finden. Daraus ergibt sich für die Prototyping Plattform die Anforderung, eine möglichst umfassende Entkopplung der zu erstellenden Anwendungen vom Lebenszyklus dieser Bestandteile zu ermöglichen.

3.2 Analyse laufender (Prototyping-)Projekte

Um diese Entkopplung zu gewährleisten wird den Entwicklern von Automotive Services ein „Komponentenbaukasten“ zur Verfügung gestellt, der sowohl gängige Schnittstellen zum Fahrzeug beinhaltet, aber auch um neue Komponenten erweitert werden kann. Damit ist es den Entwicklern möglich sich stärker auf neuartige Konzepte in Ihren Services oder die Anwendungslogik ihrer Services zu fokussieren anstatt Entwicklungszeit auf die Fahrzeugintegration zu verwenden.

Die Prototyping Plattform soll zu diesem Zweck einen Satz a priori definierter Schnittstellen zur Integration neuartiger mobiler Dienste in die Fahrzeugumgebung anbieten. Dies beinhaltet unter anderem die Bereiche der Mensch-Maschine Schnittstelle, Fahrzeugdatenschnittstellen, Telekommunikationsschnittstellen sowie Sicherheitskomponenten. Im Bereich der Mensch-Maschine Schnittstelle werden verschiedene Wege der Benutzereingabe (z.B. haptisch und akustisch) und Systemausgaben (z.B. visuell und akustisch) betrachtet. Zur Bereitstellung der Fahrzeugdatenschnittstelle wird ein Zustandsmodell des Automobils vorgehalten, mit

dessen Hilfe statische und dynamische Eigenschaften und Zustände abgebildet werden und Zustandsänderungen (Events) propagiert werden können. Im Bereich der Telekommunikationsschnittstellen werden Methoden zur Übertragung von Daten, per Rundfunk in das Fahrzeug oder per Mobilfunk sowohl in das Fahrzeug wie auch aus dem Fahrzeug, betrachtet. Sicherheitskomponenten stellen ein mögliches Mittel zur Verfügung die auf Basis der Prototyping Plattform entstehenden Prototypen hinsichtlich der Sicherheit ihres Einsatzes im Straßenverkehr zu evaluieren.

Aus dieser Grundidee ergeben sich zwei offene Punkte: zum einen muss definiert werden welche Teile der Infrastruktur eingebunden werden und zum anderen muss festgelegt werden welcher Art der Nachrichtenaustausch, bzw. die Schnittstelle, ist. Um einen ersten Eindruck über die Priorisierung bestimmter Komponenten der Fahrzeuginfrastruktur zu erhalten wurde gemeinsam mit Dienste-Entwicklern eine Matrix aufgestellt in der aktuelle Projekte und in diesen Projekten verwendete Komponenten aufgeschlüsselt werden (s. Abbildung 2). Da nicht jede Projektdokumentation offen zugänglich war erfolgte die Aufschlüsselung zum Teil anhand der vorgestellten (idealtypischen) Showcases.

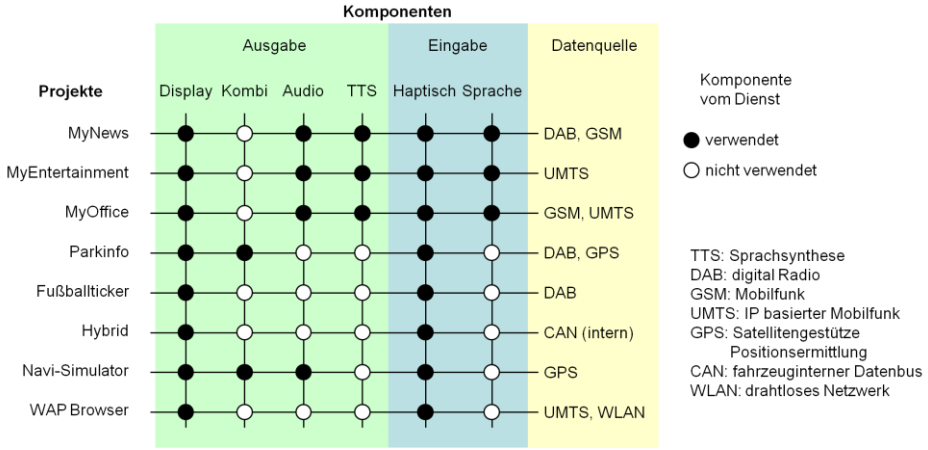

Abbildung 2: In verschiedenen Projekten verwendete Komponenten (eigene Darstellung)

4 Erste Gestaltungsempfehlungen für die Architektur der Prototyping Plattform

Im folgenden Abschnitt soll eine erste Gestaltungsempfehlung für die Architektur der Prototyping Plattform auf Basis der vorgestellten Anforderungen präsentiert werden.

Während die Anforderungen aus den Bereichen der API Gestaltung eher die Umsetzung der Schnittstellen betrifft, finden sich in den übrigen Bereichen auch Anforderungen, welche über die Architektur Beachtung finden. Das gilt z.B. für die Punkte der Anforderungen an Software, die das Automobil als Zielplattform haben. So bietet ei ne

komponentenorientierte Plattform die Möglichkeit die Komplexität der Zielplattform zu reduzieren. Außerdem ist es durch einen komponentenorientierten Ansatz möglich einzelne Bestandteile zu überarbeiten oder auszutauschen. Damit können nicht nur neue Technologien nachträglich in die Plattform integriert werden, sondern auch neue Ansätze für die Mensch-Maschine getestet werden.

Auf Basis der Analyse bestehender Projekte und den entsprechenden Anforderungen wurde das folgende Komponentenmodell für Dienste auf Basis der Prototyping Plattform entworfen. Die Bestandteile „Fahrzeugmodell“ , „Eingabe“, „Ausgabe“ sowie „Telekommunikation“ und die Unterpunkte „Rundfunk“ und „Mobilfunk“ sind dabei keine Komponenten, sondern dienen als Kategorien der Einordnung und hierarchischen Strukturierung der übrigen Bausteine.

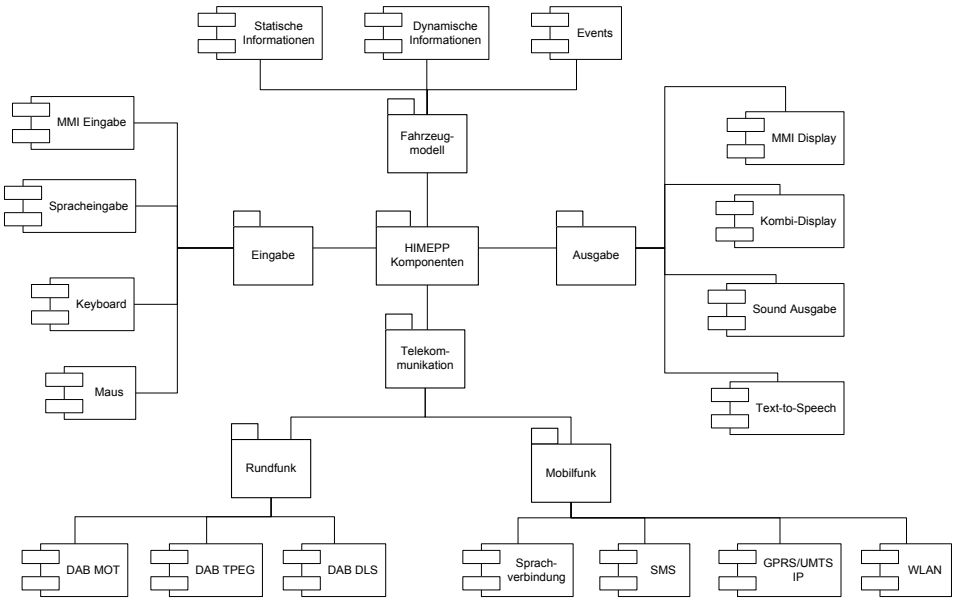

Abbildung 3: UML Komponentenmodell der Prototyping Plattform (eigene Darstellung)

Das Komponentenmodell ist dabei modular aufgebaut um Änderungen an Komponenten vornehmen zu können oder um das Modell um zusätzliche Komponenten zu erweitern, ohne dass dabei bestehende Anwendungen nicht mehr lauffähig sind. Das bedeutet, dass an einzelnen Komponenten Änderungen vorgenommen werden können, z.B. eine neue Sprachsynthese Engine angeschlossen werden kann, aber auch neue Komponenten (z.B. eine Komponente um WiMAX Verbindungen herstellen zu können) in das Komponentenmodell aufgenommen werden können. Diese Modularität ermöglicht es ebenfalls den Benutzern der Prototyping Plattform für ihre Anwendung benötigte, jedoch (noch) nicht verfügbare Komponenten selbst hinzuzufügen.

Die einzelnen Komponenten sind so aufgebaut, dass als Interface nach außen sichtbar die angebotenen Funktionsaufrufe (wenn vorhanden) und in jedem Fall eine Methode

zur Registrierung als Event Empfänger sichtbar sind. Die Funktionsaufrufe als Interface erklären sich von selbst: bietet eine Komponente anderen Komponenten eine bestimmte Funktionalität an macht sie diese so zugreifbar. Ein Beispiel ist die Komponente zur Sprachsynthese von Texten. Das Interface nach außen definiert die Methodensignatur, bestehend aus dem Methodennamen („synthesize“) und (optionalen) Übergabeparametern („String content“). Die Registrierung eines Event Listeners als Interface für die Komponenten zu definieren hat den Grund, dass es somit möglich wird anderen „interessierten“ Komponenten rasch Meldungen über den aktuellen Zustand oder ein aktuelles Ereignis zukommen zu lassen. Ein Beispiel ist das Ereignis, dass die Spracherkennung einen Sprachbefehl erkannt hat und alle als Eventlistener registrierten Komponenten darüber informiert. Eine Endanwendung kann dieses Event dann zum Anlass nehmen über das Audiointerface einen Quittungston auszugeben, damit der Nutzer des Systems weiß, dass sein Befehl erkannt wurde. Das Interface für die Registrierung ist analog aufgebaut, als Übergabeparameter dient hier die Referenz auf die sich registrierende Komponente.

5 Fazit und Ausblick

Der Einsatz einer Rapid-Prototyping-Plattform für Software im Automobil scheint ein vielversprechender Ansatz zu sein die Komplexität des Prototyping für die Entwickler zu reduzieren. Vordefinierte und bereits implementierte Komponenten, z.B. zur Mensch-Maschine-Kommunikation, erlauben es den Entwicklern neuer Dienste sich auf die wesentlichen Neuerungen in der Anwendungslogik zu fokussieren. Die offene Modulstruktur erlaubt es darüber hinaus jederzeit neue Komponenten zu dem „Baukasten“ hinzuzufügen und somit eine langfristige Nutzbarkeit des Werkzeuges sicherzustellen.

Allerdings sind noch einige zentrale Punkte offen bevor die Nützlichkeit des Ansatzes evaluiert werden kann. Für Benutzer ohne jegliche Kenntnisse im Software Engineering, die z.B. den Status ihrer neuen Hardwarekomponente zeigen oder eine Idee aus dem Marketing visualisieren möchten, wäre eine grafische Oberfläche zum codefreien Programmieren interessant. Für Benutzer mit Kenntnissen in der Programmierung von Diensten würden die für eine derartige Oberfläche nötigen Zugeständnisse an die Flexibilität der Plattform aber wohl ein erhebliches Problem darstellen. So hängen von den zukünftigen Benutzern der Plattform wesentliche Designentscheidungen ab. Da benutzerorientierte Dienste im Automobil umgesetzt werden sollen, ergeben sich erhöhte Anforderungen (timing constraints, vgl. [Saad03]) was die Mensch-Maschine-Kommunikation betrifft. Wie muss also ein komponentenbasiertes, verteiltes System ausgelegte sein, um diese Anforderungen erfüllen zu? Nicht zuletzt spielt die Möglichkeit die Plattform zu erweitern eine wesentliche Rolle. Wie lassen sich jederzeit neue Komponenten zur Plattform hinzufügen, egal ob es sich dabei um ein neues grafisches Bedienkonzept handelt das erprobt werden soll, oder einen neuen Sensor oder eine neue Kommunikationsschnittstelle? Da zur Beantwortung dieser Fragen zum größten Teil noch die Erfahrungswerte in der Praxis fehlen bietet sich die praktische Umsetzung und Evaluation der theoretisch erarbeiteten Ergebnisse gemeinsam mit einem Automobilhersteller an.

Literatur

[Bloc06] Bloch, J.: How to Design a Good API and Why it Matters. Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, Portland 2006, ACM Press, City, 2006; S. 506-507.

[Broy03] Broy, M.: Software im Automobil - Potenziale, Herausforderungen, Trends. Hrsg.):Proceedings of the 33. Jahrestagung der GI - Automotive SW Engineering & Concepts, Frankfurt/Main 2003, City, 2003.

[Bubb93] Bubb, H.: Systemergonomie - Systemergonomische Gestaltung. In: (Schmidtke, H., Hrsg.):Ergonomie, 3. Auflage, Carl Hanser, München, 1993; S. 390-420.

[DHJK94] Desmarais, M. C.; C. Hayne; S. Jagannath; R. Keller: A survey on user expectations for interface builders. Hrsg.):Conference on Human Factors in Computing Systems - Conference companion on Human factors in computing systems, ACM Press, Boston, 1994; S. 279-280.

[EAGS05] Eklund, U.; Ö. Askerdal; J. Granholm; A. Alminger; J. Axelsson: Experience of introducing reference architectures in the development of automotive electronic systems, in: ACM SIGSOFT Software Engineering Notes - Software Engineering for Automotive Systems, Vol. 30 (2005), Nummer 4, pp. 1-6.

[Fros03] Frost & Sullivan: Customer Attitudes and Perceptions Towards Telematics in Passenger Vehicles Market, 2003.

[Fuhr01] Fuhr, A.: Die Telematik ist tot - es lebe die rollende Schnittstelle. Proceedings of the Euroforum Jahrestagung "Telematik", Bonn 2001, City, 2001.

[Hart04] Hartmann, J.: Wo viel Licht ist, ist starker Schatten - Softwareentwicklung in der Automobilindustrie, in: Automatisierungstechnik, Vol. 52 (2004), Nummer 8, pp. 353-358.

[Henn07] Henning, M.: API - Design Matters, in: API Design, Vol. 5 (2007), Nummer 4, pp. 24-36.

[HiSS07] Hildisch, A.; J. Steurer; R. Stolle: HMI generation for plug-in services from semantic descriptions, *International Conference on Software Engineering - 4th Workshop on Software Engineering for Automotive Systems*, Minneapolis.

[HiHa93] Hix, D.; H. R. Hartson: Developing User Interfaces - Ensuring Usability Through Product & Process, John Wiley & Sons Inc., New York, 1993.

[HoLK07] Hoffmann, H.; J. M. Leimeister; H. Krcmar: Automotive Service Engineering - Systematische Entwicklung personalisierbarer und interaktiver mobiler Dienste für den Automobilsektor. In: (Oberweis, A.; C. Weinhardt; H. Gimpel; A. Koschmider; V. Pankratius; B. Schnizeler, Hrsg.):Proceedings of the 8. Internationale Tagung Wirtschaftsinformatik, Karlsruhe 2007, Universitätsverlag Karlsruhe, City, 2007; S. 327-344.

[HoLK07b] Hoffmann, H.; J. M. Leimeister; H. Krcmar: Pilotierung mobiler Dienste im Automobilsektor. In: (Reichwald, R.; H. Krcmar; S. Reindl, Hrsg.):Mobile Dienste im Auto der Zukunft - Konzeption, Entwicklung, Pilotierung, EUL Verlag, Lohmar, 2007; S. 125-203.

[Krais93] Kraiss, K.-F.: Systemergonomie - Mensch-Maschine-Dialog. In: (Schmidtke, H., Hrsg.):Ergonomie, 3. Auflage, Carl Hanser, München, 1993; S. 446-458.

[MeFF04] Mercer Management Consulting; Fraunhofer IPA; Fraunhofer IML (Hrsg.) (2004): Future Automotive Industry Structure (FAST) 2015 - die neue Arbeitsteilung in der Automobilindustrie Verband der Automobilindustrie, Frankfurt am Main 2004.

[Moha06] Mohan, L. R.: Driving down the Fast Lane: Increasing Automotive Opportunities the EMS Provider Way. In: http://www.frost.com/prod/servlet/market-insight-print.pag?docid=67150588, zugegriffen am: 29.04.

[Nied03] Niedermaier, F. B.: *Entwicklung und Bewertung eines Rapid-Prototyping Ansatzes zur multimodalen Mensch-Maschine-Interaktion im Kraftfahrzeug*, Technische Universität München.

[Niel93] Nielsen, J.: Usability Engineering, Academic Press, Boston, 1993.

[PBKS07] Pretschner, A.; M. Broy; I. H. Krüger; T. Stauner: Software Engineering für Automotive Systems - A Roadmap. Hrsg.):Proceedings of the 2007 International Conference on Software Engineering - Future of Software Engineering, Minneapolis 2007, IEEE Computer Society, City, 2007.

[Rühm93] Rühmann, H.: Systemergonomie - Schnittstellen in Mensch-Maschine-Systemen. In: (Schmidtke, H., Hrsg.):Ergonomie, 3. Auflage, Carl Hanser, München, 1993; S. 420-445.

[Saad03] Saad, A.: Prototyping bei der BMW Car IT GmbH, in: JavaSpektrum, (2003), Nummer 2, pp. 49-53.

[Scaf06] Scaffidi, C.: Why are APIs difficult to learn and use?, in: ACM Crossroads, Vol. 12 (2006), Nummer 4.

[Schm93] Schmidtke, H. (Hrsg.) (1993): Ergonomie (3. ed.). 3., Carl Hanser, München 1993.

[ShRP07] Sharp, H.; Y. Rogers; J. Preece: Interaction Design, 2nd. 2nd, Wiley, Chichester, 2007.

[Werd05] Werder, H.: Verkehrstelematik als Element der Verkehrspolitik. Proceedings of the its-ch, Olten 2005, City, 2005.

[WiHe07] Williams, M.; R. Helbig: Probandenstudie - Evaluierung des Dienstes MACS MyNews. In: (Reichwald, R.; H. Krcmar; S. Reindl, Hrsg.):Mobile Dienste im Auto der Zukunft, EUL Verlag, Lohmar, 2007; S. 205-272.

[ZCTT05] Zhang, P.; J. Carey; D. Te'eni; M. Tremaine: Integrating Human-Computer Interaction Development into the Systems Development Life Cycle: A Methodology, in: Communications of Association for Information Systems, Vol. 15 (2005), pp. 512-543.