Daniel Burgstahler, Stefan Schulte, Sven Abels, Kristof Kipp, Philipp Hoenisch, Scharam Dustdar and Ralf Steinmetz: Informationssysteme für Verkehrsteilnehmer: Datenintegration, Cloud-Dienste und der Persönliche Mobilitätsassistent. In: PIK - Praxis der Informationsverarbeitung und Kommunikation, vol. 37, p. 243-250, September 2014. ISSN 0930-5157.

Autoren: Daniel Burgstahler[1], Stefan Schulte[2], Sven Abels[3], Kristof Kipp[3], Philipp Hoenisch[2], Schahram Dustdar[2], Ralf Steinmetz[1]

Informationssysteme für Verkehrsteilnehmer: Datenintegration, Cloud-Dienste und der Persönliche Mobilitätsassistent

Zusammenfassung

Verkehrsteilnehmern steht heute theoretisch eine Vielzahl von Informationsquellen zur Verfügung. Praktisch gesehen gestaltet sich die Nutzung dieser Informationen für den Endnutzer jedoch schwierig, denn diese liegen in verschiedenen Apps, Softwarediensten oder in Form anderer Datenquellen vor. Eine einheitliche Sicht auf für Verkehrsteilnehmer interessante Informationen wird somit erschwert oder gar unmöglich.

In diesem Beitrag werden verschiedene Forschungsansätze im Bereich von Informationssystemen für Verkehrsteilnehmer vorgestellt, welche es Softwareentwicklern erleichtern sollen, entsprechende Informationen dem Endnutzer zur Verfügung zu stellen. Dazu gehören Datenintegration, die Verwendung mobilitätsrelevanter Daten in Cloud-Diensten und letztendlich die Integration von Cloud-Diensten in einen Persönlichen Mobilitätsassistenten, welcher eine multimodale Benutzerschnittstelle anbietet.

Motivation

Mit der zunehmenden Integration von Informations- und Kommunikationstechnologien in Fahrzeugen sowie der persönlichen Vernetzung durch die Nutzung von Smartphones und anderen Endnutzergeräten steht Verkehrsteilnehmern bereits heute eine Vielzahl von Datenquellen zur Verfügung. Nicht nur in technischer, sondern auch in inhaltlicher Hinsicht sind diese Datenquellen überaus vielfältig. Sie reichen von Online-Navigationssystemen über Echtzeit-Verkehrssensoren und Fahrzeugsensoren, (derzeit noch prototypischen) Car-to-X-Systemen, Daten zur Bestimmung der Position und Ankunftszeit von Straßenbahnen und Bussen bis hin zu historischen Daten in Form von *Open Government Data*, d. h. von öffentlichen Institutionen frei zugänglich gemachten Daten wie beispielsweise Unfallstatistiken [12, 14].

Der Zugriff auf diese Datenquellen wird in der Regel durch Apps realisiert, welche entweder via Smartphone oder mittels in einem Fahrzeug eingebautem Infotainmentsystem (bspw. BMW ConnectedDrive, Volkswagen CarInfotainment oder FIAT Blue&Me) dem Verkehrsteilnehmer zur Verfügung gestellt werden. Das Auto wird somit zum mobilen Endgerät [4]. Trotz der Bestrebungen der (Automobil-) Industrie lässt sich jedoch feststellen, dass bestehende Apps die Möglichkeiten von Datenintegration aus unterschiedlichen Quellen nicht nutzen und häufig von eher informativer Natur sind [1]. Infotainmentsysteme bieten zudem häufig weder die Möglichkeit einer Personalisierung noch eine Erweiterbarkeit wie sie beispielsweise durch moderne App-Verkaufsportale in Smartphones zu finden ist. Weiterhin finden sich in der Regel keine Ansätze zur Integration von

[1] Multimedia Communications Lab, Technische Universität Darmstadt
[2] Distributed Systems Group, Technische Universität Wien
[3] Ascora GmbH, Ganderkesee

Mobilitätsfunktionalitäten – stattdessen werden einzelne Datenquellen durch individuelle Apps zur Verfügung gestellt, mit denen der Nutzer wiederum einzeln interagieren muss.

Dies führt zu einem unnötig hohen Aufwand bei der Informationsbeschaffung. Insbesondere bei der Verwendung von Mobilitäts-Apps als Fahrzeugführer kann dies zu kritischen und gefährlichen Situationen sowohl für App-Nutzer als auch andere Verkehrsteilnehmer führen. Es ist daher wenig überraschend, dass die Nutzung von Smartphone-Apps durch Autofahrer während der Fahrt momentan in Deutschland nicht erlaubt ist – dies schließt im Übrigen auch die Nutzung des Smartphones als Navigationsgerät ein [18]. Die Vielzahl einzelner Apps führt weiterhin dazu, dass Nutzer umständlich Daten an unterschiedlichen Stellen zusammensuchen und häufig mittels Texteingabe übertragen müssen. Selbst relativ einfache Funktionalitäten wie die Verknüpfung von Kalendereinträgen und Navigationssystemen sind heute nicht standardmäßig verfügbar.

Statt einer Vielzahl von verschiedenen Mobilitäts-Apps wäre den Verkehrsteilnehmen mehr geholfen, wenn alle mobilitätsbezogenen Funktionen gebündelt angeboten würden. Ein solcher *Persönlicher Mobilitätsassistent (PMA)* sollte unter anderem die folgenden Erfordernisse erfüllen [10]:

- Integrierte Lösung: Der PMA sollte wichtige Informationen und Funktionalitäten bündeln und erweiterbar sein. Diese Funktionalitäten können wie Apps in den PMA integriert werden, sind jedoch immer Teil des PMAs und damit einer ganzheitlichen Lösung.
- Sichere, ablenkungsfreie Bedienbarkeit: Im Mobilitätskontext ist die einfache und sichere Bedienbarkeit von immanenter Wichtigkeit. Da ein PMA in unterschiedlichen Situationen genutzt werden kann, bspw. im Auto aber auch in der Bahn, sollte eine multimodale Bedienung möglich sein: Während im Auto eine Bedienung via Tastatur vermieden werden sollte, bietet sich eine Sprach- oder Gestensteuerung bzw. eine Kombination daraus an. In der Bahn hingegen könnte eine Steuerung durch Sprache oder Gesten von Mitreisenden als störend empfunden werden.

Automobilhersteller haben bereits erkannt, dass mobile Apps und Dienste in Zukunft einen wichtigen Teil des „Produkts“ Mobilität darstellt [7]. Die „App-Revolution“ im Handymarkt hat jedoch auch gezeigt, dass viele innovative Apps von Drittanbietern zur Verfügung gestellt werden – gleiches gilt auch für den Markt für Mobilitäts-Apps [5]. Der Erfolg eines PMAs ist entsprechend nur gewährleistet, wenn diese Drittanbieter bei der Entwicklung von innovativen Mobilitäts-Apps unterstützt werden.

Wie bereits zuvor erwähnt, ist ein Problem bei der Entwicklung von Mobilitäts-Software die Vielzahl heterogener Datenquellen, welche zur Verfügung stehen. Selbst bei einer Analyse ähnlicher Datenquellen im Bereich Mobilität lässt sich feststellen, dass die Datenformate häufig völlig unterschiedlich sind. Betrachtet man beispielsweise Bikesharing-Anbieter in Europa, so bieten viele Städte Echtzeit-Informationen über die zur Verfügung stehende Anzahl an Fahrrädern an verschiedenen Standorten innerhalb der jeweiligen Stadt an. In den meisten Fällen geschieht dies mittels der Darstellung auf einer Karte [2], einige Anbieter wie bspw. JCDecaux[4], welche Bikesharing z. B. in Brüssel, Paris und Santander betreibt, bieten jedoch auch APIs und somit Dateiformate wie CSV oder JSON an. Häufig werden die Daten auch von Drittanbietern unentgeltlich aus der Karten-basierten Darstellung extrahiert und in einem anderen Format zur Verfügung gestellt, bspw. für die

[4] https://developer.jcdecaux.com/#/opendata/vls?page=getstarted

Stadt London[5] im XML-, JSON-, CSV- und YAML-Format, oder von CityBikes[6] im JSON-Format für eine Vielzahl von Orten auf der ganzen Welt. Da kein Beschreibungsstandard für Echtzeit-Bikesharing-Daten existiert, ist es wenig überraschend, dass die Datenströme sich stark unterscheiden – möchte ein Entwickler also Apps mit Livedaten anbieten, so ist eine manuelle Anpassung des Datenformats notwendig. Ein gemeinsames Datenmodell würde hingegen den Entwicklungsaufwand minimieren und – statt einer Vielzahl von Apps für unterschiedliche Städte – eine einheitliche App für diese Daten ermöglichen.

Natürlich handelt es sich bei den Bikesharing-Datenquellen nur um ein exemplarisches Beispiel. Dennoch lässt sich die beobachtete Heterogenität an Datenformaten und -protokollen auch bei anderen mobilitätsrelevanten Datenquellen erkennen. Statt sich mit vielen unterschiedlichen Datenformaten und Kommunikationsprotokollen auseinandersetzen zu müssen, sollten Entwickler daher durch ein einheitliches Datenmodell und die Möglichkeit, Datenquellen anderer Formate automatisiert zu transformieren, unterstützt werden. Dabei sollte dem *Data-as-a-Service*- (DaaS-) Prinzip gefolgt werden, d. h. Daten sollten unabhängig von ihrem Standort und der verwendeten Technologien über einheitliche Softwareschnittstellen zur Verfügung gestellt werden [13]. Weiterhin sind heute nur wenige Softwareentwickler mit der Entwicklung multimodaler Benutzerschnittstellen vertraut und müssen entsprechend durch APIs, Softwarewerkzeuge und Dokumentation unterstützt werden.

Um der Realisierung des Persönlichen Mobilitätsassistenten näher zu kommen, untersucht das EU-Projekt „SIMPLI-CITY – The Road User Information System of the Future“[7] neue Technologien und Funktionalitäten in den folgenden Teilbereichen: (i) (Mobilitäts-) DaaS, (ii) eine Softwareplattform für Mobilitäts-Apps und -Dienste, (iii) mobile, multimodale Benutzerschnittstellen und (iv) Entwicklerunterstützung durch entsprechende Softwarewerkzeuge.

Im weiteren Verlauf dieses Berichts werden diese Beiträge anhand der SIMPLI-CITY-Gesamtarchitektur sowie ausgewählter Forschungsfragestellungen aus den Bereichen DaaS sowie Dienst- und App-Plattform erläutert. Weiterhin werden verwandte Forschungsarbeiten und kommerzielle Lösungen bewertet. Der Bericht endet mit einer kurzen Zusammenfassung und einem Ausblick auf offene Fragestellungen.

SIMPLI-CITY-Gesamtarchitektur

Um die oben genannten vier maßgeblichen Funktionalitäten einer ganzheitlichen PMA-Lösung zu erfüllen, ist das SIMPLI-CITY-Softwareframework in vier maßgebliche Bereiche aufgeteilt, welche in Abbildung 1 zu sehen sind. Innerhalb des PMAs werden Apps ausgeführt, die direkt oder indirekt mit dem Nutzer interagieren. Apps werden dabei multimodal realisiert und lassen sich demnach sowohl per Fingerzeig als auch per Sprache steuern.

Den ersten Bereich der Architektur stellen Komponenten dar, welche den eigentlichen PMA bilden, d. h. solche Softwarekomponenten, welche auf einem mobilen Gerät die Ausführungsumgebung für Apps und die Benutzerschnittstelle zur Verfügung stellen (in der Abbildung lila dargestellt). Das mobile Gerät kann dabei ein handelsübliches Smartphone bspw. auf Android-Basis darstellen. Eine Möglichkeit zur Verbindung mit dem Fahrzeug ist ein OBD II-Adapter welcher per Bluetooth mit dem

[5] http://bike-stats.co.uk/
[6] www.citybik.es
[7] http://www.simpli-city.eu

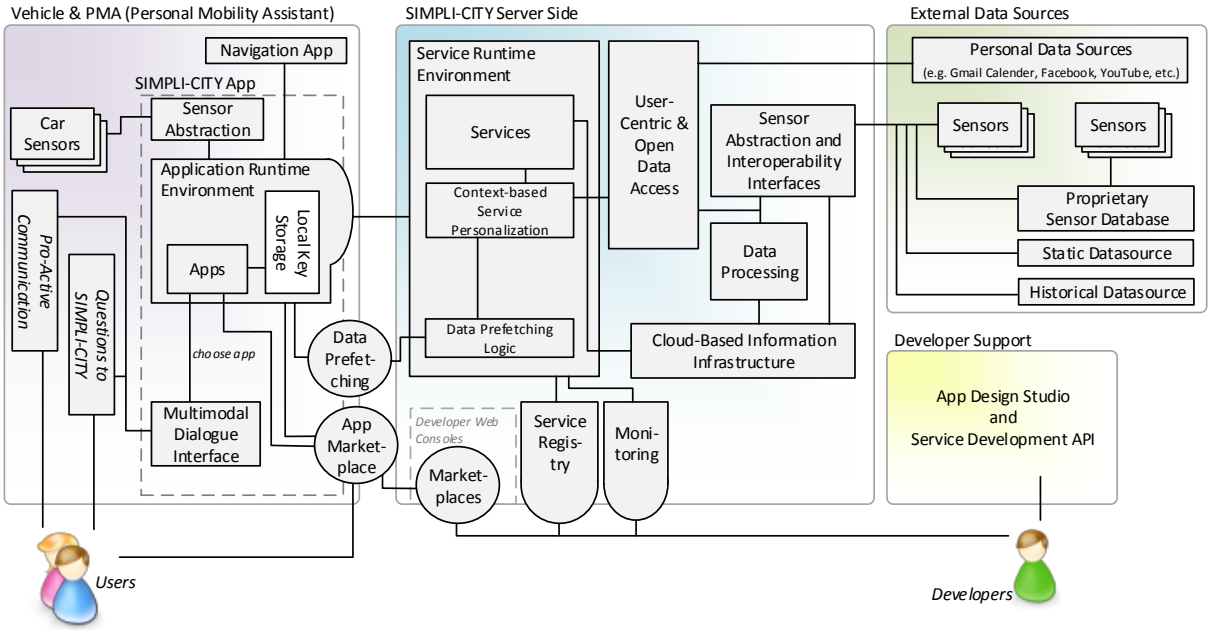

Abbildung 1: SIMPLI-CITY-Gesamtarchitektur

Smartphone verbunden wird und somit Zugriff auf einige der Sensor-Informationen des Fahrzeuges ermöglicht. Sowohl auf Seiten führender Automobilhersteller als auch der führenden Anbieter von Smartphone-Betriebssystemen ist das Bestreben der Integration von Fahrzeug und Smartphone inzwischen sehr groß [11], so dass es in Neufahrzeugen möglich sein wird, ein Smartphone direkt mit dem Fahrzeug zu verbinden oder eine entsprechende Anwendung direkt auf dem Infotainmentsystems des Fahrzeugs zu installieren (vgl. auch Abschnitt zu Verwandten Arbeiten).

Der zweite Bereich der Architektur beinhaltet die Serverseite der SIMPLI-CITY-Plattform (in der Abbildung blau dargestellt). Der PMA folgt dabei dem Software-as-a-Service (SaaS)-Prinzip, d. h. Softwarefunktionalitäten werden zunächst einmal über die Cloud zur Verfügung gestellt und von PMA-Apps über ein mobile Datenverbindung genutzt. Die Apps bieten somit die relativ leichtgewichtige Funktionalität eines *Thin Clients* sowie die Benutzerschnittstelle, während Cloud-Dienste die eigentliche Softwarelogik und Anbindung an nicht-lokale Datenquellen anbieten.

Der PMA soll den Zugriff auf verschiedene Informationsquellen wie bspw. auf Daten externer (Verkehrs-)Sensoren oder auf Daten des persönlichen Kalenders erlauben. Diese Datenquellen und zugehörige Zugriffsmethoden bilden den dritten Bereich des SIMPLI-CITY-Softwareframeworks (in der Abbildung grün dargestellt). Der Zugriff wird dabei über eine Datenabstraktion ermöglicht, die eine einheitliche Zugriffsschicht auf unterschiedliche Datenquellen darstellt und somit DaaS basierend auf einem einheitlichen Datenmodell realisiert. Bisher vorgesehen ist der Zugriff auf Sensordaten, Mediendaten, Nutzerprofildaten und Daten aus Open (Government) Data Repositories.

Die Grundidee des PMAs ist die Erweiterung der angebotenen Funktionalitäten durch die Integration neuer Apps. Um dies zu erreichen, verfolgt SIMPLI-CITY einen offenen Ansatz bei dem App- und Dienst-Entwickler neue Funktionen bereitstellen können, die über einen Marktplatz kostenfrei oder kostenpflichtig angeboten werden. Der Entwicklungsbereich stellt deshalb den vierten Block des SIMPLI-CITY Softwareframeworks dar und enthält neben Dokumentationen, Beispielen und APIs auch eine IDE bei der Plugins die einfache Entwicklung von Apps und Dienste unterstützen (in Abbildung 1 gelb dargestellt).

Die folgenden Abschnitte beschreiben einzelne Aspekte der obigen Architektur im Detail, insbesondere die Anbindung von Datendiensten (DaaS) sowie das Konzept der Dienst- und App-Plattform.

Data-as-a-Service

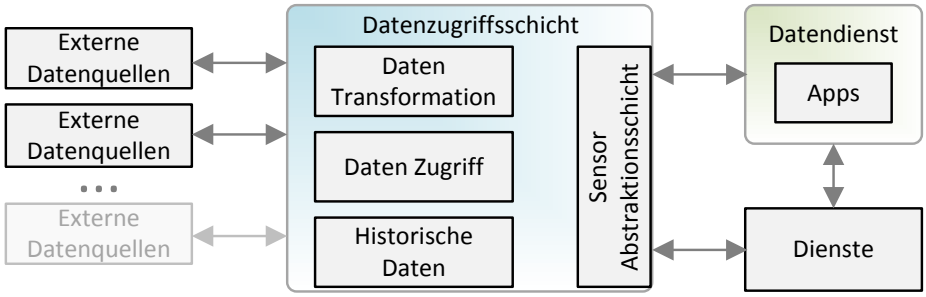

Abbildung 2: Data-as-a-Service – Sensor-Abstraktionsschicht

Um einen einfachen und einheitlichen Zugriff auf heterogene Datenquellen zu ermöglichen, ist die nahtlose Integration und die Bereitstellung von mobilitätsrelevanten Daten als einheitlicher Dienst eine wesentliche Voraussetzung für die Realisierung des PMA. Momentan unterstützt werden Datenströme, der Direktzugriff auf einzelne Sensorwerte und historisierte oder statische Daten. Statische Mobilitätsdaten sind in einer Datenbank abgelegt oder anderweitig verfügbar und ändern sich relativ selten; Beispiele hierfür sind Informationen über Geschwindigkeitsbeschränkungen, Busfahrpläne, oder (Open-Data-) Verkehrsstatistiken. Diese sind meist für einen längeren Zeitraum valide, sind aber durchaus Veränderungen unterworfen. Mobilitätsbezogene Datenströme sind ebenfalls vielfältig und umfassen beispielsweise Sensoren zur Verkehrsflussmessung, Ampelschaltungen, oder Informationen über freie Parkplätze. Die eingesetzten Technologien reichen von Induktionsschleifen oder Drucksensoren bis hin zu Kamerasystemen [8]. Im einfachsten Falle werden bereits aufbereitete Daten zur Verfügung gestellt, wie der Verkehrsfluss in Fahrzeugen je Zeitraum oder die freien Parkplätze in einem bestimmten Parkhaus, es ist jedoch auch eine auf Sensorik basierende feinere Granularität der Daten möglich.

Der Zugriff auf die verschiedenen heterogenen Datenquellen wird in SIMPLI-CITY durch eine Dienstschicht bereitgestellt, welche einen einheitlichen Zugriff ermöglicht. Eine vereinfachte Übersicht dieser Datenzugriffsschicht ist in Abbildung 2 zu sehen. Über einheitliche Schnittstellen können sowohl PMA-Apps als auch Cloud-Dienste auf externe Informationen zugreifen und erhalten die angefragten Daten in einem einheitlichen Datenformat zurück.

Um Mobilitätsdaten als Dienst bereitstellen zu können, kann die abgebildete Abstraktionsschicht über verschiedene Zugriffsverfahren auf externe Daten zugreifen und diese automatisiert in ein einheitliches Datenformat übersetzten. Die angebotenen Zugriffsverfahren umfassen das Abonnieren von Daten über Publish-Subscribe-Mechanismen, den direkten Zugriff, um ein einzelnes Datum zu erhalten, aber auch das regelmäßige Abfragen einer Datenquelle. Alle Daten können historisiert und somit für eine spätere Verwendung oder Analyse bereitgestellt werden um z. B. aus dem Verlauf der Datenwerte weitere Informationen ableiten zu können. Ein solches Verfahren kann zum Beispiel bei der Vorhersage von Stauinformationen Anwendung finden.

Eine wichtige Komponente stellt dabei das mobile Endgerät des Benutzers dar. Einerseits dient das Endgerät als Ausführungsplattform für PMA-Apps, andererseits ist es aber auch eine wichtige

Komponente zur Datenerfassung. Sowohl interne Sensoren als auch benutzerbezogene Daten können hierüber nicht nur direkt in den Apps verwendet werden sondern auch von den Cloud-Diensten. Die Basis für den Datenzugriff stellt hierbei ein Hintergrunddienst auf dem mobilen Endgerät dar. Dieser stellt eine einheitliche Schnittstelle zu allen lokalen Sensoren bereit. Zu den lokalen Sensoren gehören nicht nur die in das Smartphone integrierten Sensoren sondern auch Sensoren von verbundenen Geräten. Dieser Hintergrunddienst ist unter anderem für die Kommunikation mit dem Fahrzeug zuständig, was ein Einbinden der Fahrzeugsensoren ermöglicht, welche dann ebenfalls als lokale Sensoren bereitgestellt werden. Somit können Apps über die gleiche Schnittstelle auf alle lokalen Sensoren zugreifen und erhalten die entsprechenden Sensorwerte in einem einheitlichen Datenformat. Der lokale Hintergrunddienst kommuniziert bidirektional mit einer komplementären Komponente auf der Serverseite und stellt den Zugriff auf die verfügbaren Sensoren bereit. Jeder Sensor ist im System mit einer eindeutigen ID versehen. Über die serverseitig bereitgestellten Schnittstellen können somit auch lokale Sensoren des PMA oder des angebundenen Fahrzeugs abgefragt werden. Sensordaten werden somit auch an die SIMPLI-CITY-Gesamtarchitektur angebunden und sind somit für Cloud-Dienste aufruf- und verwendbar.

Softwareplattform für Mobilitäts-Apps und -Dienste

Wie bereits in der Übersicht des Softwareframeworks beschrieben, trennt SIMPLI-CITY Softwarefunktionalitäten in Cloud-Dienste, welche SaaS anbieten, und die PMA-Apps, welche nur eine leichtgewichtige Funktionalität implementieren und somit den Grundprinzipien eines *Thin Clients* folgen. Vorteile dieses Ansatzes sind die Wiederverwendbarkeit der Cloud-Dienste in verschiedenen Apps, serverseitige Dienstkomposition, die lose Kopplung von Apps und Diensten, so dass ein Dienst leicht ausgetauscht werden kann, sowie die Möglichkeit, Dienstaufrufe über das Softwareframework abrechnen und überwachen zu können.

Die eigentliche Dienstausführungsumgebung (engl. „Service Runtime Environment“) basiert auf der OSGi-Spezifikation in Version R4 [15]. Diese Spezifikation definiert ein modulares Komponentenmodell für Java, welches eine solide Grundlage für die im vorgestellten Szenario benötigten Funktionalitäten bietet. Insbesondere unterstützt das OSGi-Framework die Registrierung und Instanziierung von Diensten als eigenständige Module und bietet eine vollständige Versionierungslösung, d. h. verschiedene Versionen des gleichen Dienstes können ohne großen Administrationsaufwand verwendet werden. Cloud-Dienste und auch die im letzten Kapitel vorgestellten Datendienste lassen sich somit innerhalb des OSGi-Frameworks registrieren und instanziieren und stehen somit für die Nutzung in PMA-Apps zur Verfügung.

Generell ist die OSGI-Spezifikation nicht auf eine bestimmte Anwendungsdomäne zugeschnitten. Dies ermöglicht ihren Einsatz in verschiedenen Bereichen, führt jedoch auch dazu, dass für die in diesem Beitrag betrachtete Mobilitätsdomäne benötigte Funktionalitäten bisher nicht unterstützt werden. Aus diesem Grund werden verschiedene Erweiterungen benötigt, die im Folgenden kurz vorgestellt werden: die Unterstützung von Push- und Kontext-Funktionalitäten sowie die Erweiterung des OSGi-Frameworks um einen REST-Proxy.

Die Integration von *Push*-Funktionalitäten ermöglicht es, den PMA-Nutzer mit Informationsupdates zu versorgen, ohne dass der PMA einen Dienst erneut aufrufen („pullen“) muss. Dies ist insbesondere in Mobilitätsszenarien von Interesse, um bspw. Aktualisierungen über die Verkehrssituation zu erhalten ohne wiederholt einen Dienst aufrufen zu müssen. Ein großer Vorteil des Push-Ansatzes ist dabei die unmittelbare Verfügbarkeit der aktualisierten Daten in der App, wobei jedoch festzuhalten

ist, dass der Pull-Ansatz im Regelfall zu einem geringeren Energieverbrauch führt, Pushing also nicht für jeden Einsatzzweck geeignet ist [16]. Per se erlaubt das OSGi-Framework kein Pushing. Daher wird diese Funktionalität in SIMPLI-CITY durch eine entsprechende Erweiterung umgesetzt. Diese erlaubt es, dass sich an bestimmten Informationsupdates interessierte Apps bei Cloud-Diensten anmelden. Anschließend können diese aktiven, statusbehafteten Dienste eine entsprechende Aktualisierung an PMA-Apps senden, wobei die Verwaltung der Abonnements von der Push-Erweiterung übernommen wird.

Eine weitere wichtige Erweiterung stellt die Unterstützung von *Kontextdaten* dar [17]. In Mobilitätsszenarien ist insbesondere der Nutzerstandort eine Kontextinformation, welche für die Anpassung des Dienst- und damit des App-Outputs genutzt werden kann. Ein naheliegendes Beispiel ist der Austausch eines von einer App aufgerufenen Dienstes zur Laufzeit – dies wird durch die lose Kopplung von Diensten und Apps realisierbar. So ist es bspw. möglich, die gleiche Bikesharing-App in verschiedenen Städten einzusetzen, diese jedoch standortbezogene Dienste aufrufen zu lassen, um immer die am Standort relevanten Daten zu erhalten. Dies geschieht für den Nutzer transparent.

Ein weiteres Beispiel für kontextbezogene Dienstausführung ist das sogenannten *Data Prefetching*. Insbesondere mobile Nutzer sind auf die ständige Verfügbarkeit einer Datenverbindung angewiesen. In manchen Fällen ist eine solche Datenverbindung jedoch nicht gegeben, bspw. weil das monatliche Datenvolumen bereits aufgebraucht wurde, oder weil der Nutzer sich im Ausland befindet, jedoch keine Roaming-Gebühren zahlen möchte. In diesem Fall bietet es sich an, Daten schon vorab in einen lokalen Speicher zu laden, um über diese zum benötigten Zeitpunkt verfügen zu können. Dieses Prefetching geschieht vorzugsweise solange der Nutzer Daten kostenlos herunterladen kann, beispielsweise in einem Hotel-WLAN. Zur Entscheidung ob ein Prefetching durchgeführt wird, können vielfältige Kontextdaten herangezogen werden – insbesondere natürlich der (jetzige und zukünftige) Nutzerstandort oder das Nutzerprofil inkl. Daten über den Mobilfunkvertrag. Es sollte beachtet werden, dass ein Prefetching nicht für alle Daten sinnvoll ist – die Entwicklung intelligenter Prefetching-Steuerungsmechanismen ist daher ein Teil unserer zukünftigen Arbeiten.

Um den PMA als Thin Client realisieren zu können, sollte es möglich sein, Cloud-Dienste aus einer beliebigen PMA-App aufrufen zu können. Jeder dieser Dienstaufrufe sollte zudem überwacht werden. Damit diese Funktionalität nicht von jedem Dienst individuell angeboten werden muss, bietet es sich an, eine einheitliche Schnittstelle zur Verfügung zu stellen. Diese Schnittstelle kann als *Proxy* gesehen werden, der den eigentlichen Dienstaufruf kapselt, und ist somit in der Lage zusätzliche Funktionalitäten (wie Überwachung, Dienstabrechnung, Logging, ...) während eines Aufrufs nahtlos einzubinden. Weiterhin stellen OSGI-Dienste eine *Remote*-Ausführung standardmäßig nicht zur Verfügung. Daher wurde ein entsprechender *REST-Proxy* entwickelt. Dieser ermöglicht den Aufruf aller in der Dienstausführungsumgebung befindlichen Dienste über eine einheitliche Schnittstelle und reduziert somit den Aufwand für Softwareentwickler – diese müssen Überwachungs- oder Abrechnungsfunktionalitäten nicht mehr in ihre Dienste integrieren, sondern können die entsprechenden Funktionalitäten der Softwareplattform nutzen.

Neben der serverseitigen Dienstausführungsumgebung stellt die Appausführungsumgebung den clientseitigen Teil und damit die Grundlage für den PMA dar. Diese stellt die erwähnte multimodale Benutzerschnittstelle bereit und ist für Wartung und Ausführung der PMA-Apps verantwortlich. Als Basis dieser Umgebung wird Android als Betriebssystem genutzt und um ein Framework erweitert, welches die verschiedenen clientseitigen Komponenten (Aufruf von Cloud-Diensten, Push-

Nachrichten-Empfang und -Weiterleitung) miteinander verbindet und für die Apps bereitstellt. Wiederum verringert dies den Implementierungsaufwand, da wichtige Funktionalitäten zentral zur Verfügung gestellt werden und somit von Softwareentwicklern mittels API-Funktionalitäten verwendet werden können.

Als Grundlage für die multimodale Benutzerschnittstelle dient das *CTH Grammatical Framework* [6]. Die interpretierten oralen Kommandos werden über eine Liste von semantischen Begriffen einer App zugeordnet und von der Ausführungsumgebung weitergeleitet. Kontextinformationen werden hierzu vom CTH Grammatical Framework interpretiert und an die zuständige App weitergegeben. Der PMA bietet sowohl eine multimodale Eingabe als auch Ausgabe – entsprechend erfolgt die Ausgabe der PMA-Apps durch die Android-interne Sprachausgabe oder über Bildschirmelemente.

Verwandte Arbeiten

Während bisher keine ganzheitlichen Ansätze zur Erstellung eines PMAs inkl. der hier vorgestellten serverseitigen Funktionalitäten und Komponenten existieren, gibt es durchaus Lösungen, welche einzelne der hier vorgestellten Forschungsfragestellungen untersuchen [5]. So stellen Malgiocchetti et al. [3] einen PMA vor, welcher sich jedoch auf eine bestimmte Funktionalität, d. h die Auswahl nachhaltiger Beförderungsmöglichkeiten, beschränkt. Dieser PMA wurde als mobile App umgesetzt, ist jedoch nicht um weitere Apps erweiterbar. Da dieser PMA stark funktionsgetrieben ist, werden außerdem keine Forschungsfragen aus dem Bereich Benutzerschnittstellen oder Datenanbindung betrachtet.

Darüber hinaus versucht die Automobilindustrie, das Auto als mobiles Endgerät zu positionieren, sei es durch die Integration von Smartphones in das Fahrzeug, oder durch den Einbau von durch Apps erweiterbaren Infotainmentsystemen.[8] Entsprechende Lösungen sind in vielen Fällen anbieterspezifisch und damit proprietär. In den meisten Fällen werden Apps nicht von Drittanbietern, sondern ausschließlich über den Automobilhersteller angeboten und beschränken sich auf Infotainment-Funktionalitäten. Eine Ausnahme stellen General Motors[9] (GM) und Ford Motors[10] dar, welche es Drittanbietern erlauben, Apps nach einer entsprechenden Sicherheitsüberprüfung im Marktplatz anzubieten. Dabei wählen die beiden Hersteller komplett gegensätzliche Ansätze. Während bei GM die Apps direkt auf dem ins Fahrzeug integrierten Infotainmentsystem ausgeführt werden, wird bei Ford die Integration von Smartphone-Apps angestrebt. Diese Smartphone-Apps werden dann über Bedienelemente des Fahrzeugs gesteuert und können Informationen vom Fahrzeug erhalten.

Ein ähnliches Ziel hat das GENIVI-Konsortium[11], welches inzwischen über 160 Partner umfasst. Ziel ist eine offene und weltweit einheitliche auf Linux basierte Softwareplattform zur Verwendung in der gesamten Automobilindustrie zu schaffen. Allerdings ist GENIVI selbst kein Produkt sondern ein Bündnis mit dem Ziel Standards zu etablieren. Dazu wurde eine Linux-basierte Referenzplattform definiert. Auf dieser Basis sind konkrete Produkte angekündigt, bspw. von Continental[12], die auch durch Drittanbieter erweiterbar sein sollen. Diese Produkte sind jedoch noch nicht verfügbar, ein konkreter Marktplatz zur Platzierung von Drittanbieter Apps noch nicht konkretisiert und viele der

[8] Eine Übersicht über existierende Ansätze vieler Hersteller findet sich auf http://simpli-city.eu/state-of-the-art.
[9] https://developer.gm.com/index.php
[10] https://developer.ford.com
[11] http://www.genivi.org
[12] http://www.automobilwoche.de/article/20120110/NACHRICHTEN/120119998/continental-bringt-2014-multimediaplattform-nach-genivi-standards#.UuYEYHlZAlI

Anbieter haben parallel auch andere proprietäre Produkte im Produktprogramm, bzw. angekündigt, so dass eine Marketablierung nicht wirklich sicher ist. Im Zuge der Google Automotive Alliance[13], welche 2014 vorgestellt wurde, hat Google angekündigt, Android auf den Automobilbereich zuschneiden und eng mit Herstellern zusammenarbeiten zu wollen. Erste Systeme wurden für 2014 angekündigt. Mirrorlink[14] und Apple CarPlay[15] stellen Interoperabilitätsstandards dar, mit denen Smartphone-Apps auf einem Infotainmentsystem genutzt werden können. Grundsätzlich nutzt Mirrorlink dabei ein Infotainmentsystem als Benutzerschnittstelle zu auf einem Android-Smartphone laufenden Apps, während CarPlay ausgewählte iOS-Funktionalitäten zur Verfügung stellt.

Tizen IVI[16]*,* das als quelloffene und HTML5-basierte Plattform für Infotainmentsysteme konzipiert ist, ist von allen hier vorgestellten Lösungen dem SIMPLI-CITY PMA funktional am nächsten. Allerdings steht bei Tizen IVI nicht das Smartphone im Mittelpunkt, sondern das im Fahrzeug vorhandene System; entsprechend sind mögliche Apps auf solche Funktionalitäten beschränkt, welche im Auto genutzt werden können, während der SIMPLI-CITY PMA auch auf mobilitätsbezogene Daten für andere Verkehrsteilnehmer abzielt.

Zusammenfassung und Ausblick

Im vorliegenden Beitrag wurden verschiedene Forschungsbeiträge aus dem Projekt SIMPLI-CITY vorgestellt. SIMPLI-CITY beschäftigt sich mit Informationssystemen für Verkehrsteilnehmer und insbesondere mit server- und clientseitigen Funktionalitäten zur Datenintegration, Dienstausführung und Benutzerinteraktion via PMA. Ein Hauptaugenmerk liegt auf der Unterstützung von Softwareentwicklern durch API-Funktionalitäten, so dass sich die Entwickler auf Kernfunktionalitäten von Diensten und Apps konzentrieren können. Die entsprechenden Softwareprototypen werden quelloffen auf der Projekthomepage angeboten und bilden somit die Grundlage für eine Alternative zu proprietären und kommerziellen Lösungen. Während die grundlegenden Softwarefunktionalitäten bereits implementiert wurden, beschäftigt sich SIMPLI-CITY momentan mit der Integration weiterer Funktionalitäten und der Umsetzung von Testszenarien in Dublin und Bologna.

Wie auch im Rahmen der *European Innovation Partnership on Smart Cities and Communities*[17] erkannt wurde, sollten die Themenfelder Energie (Smart Grids) und Mobilität zusammen betrachtet werden, um die „smarten" Gemeinschaften (engl. „Smart Communities") und Städte (engl. „Smart Cities") der Zukunft zu realisieren. Intelligente Informationssysteme für Verkehrsteilnehmer sowie die dazu beitragenden Technologien – insbesondere die hier vorgestellten Forschungsarbeiten in den Bereichen Datenintegration und Dienstausführungsumgebung – können hierfür einen ersten grundlegenden Schritt darstellen.

Danksagung

Die in diesem Artikel vorgestellten Forschungsergebnisse entstanden im Rahmen des FP7-ICT-Projekts SIMPLI-CITY, welches von der Europäischen Union gefördert wird (Grant Agreement No. 318201).

[13] http://www.openautoalliance.net/#about
[14] http://www.mirrorlink.com/
[15] http://www.apple.com/ios/carplay/
[16] https://wiki.tizen.org/wiki/IVI
[17] http://ec.europa.eu/eip/smartcities/

Literatur[18]

[1] C. Linnhoff-Popien und S. Verclas, „Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse," in Smart Mobile Apps, Springer Heidelberg Dordrecht London New York, 2012, Kap. 1, S. 3-16.

[2] O. O'Brien, J. Cheshire und M. Batty, „Mining bicycle sharing data for generating insights into sustainable transport systems," in *Journal of Transport Geography*, vol. 34, S. 262-273, 2014.

[3] D. Magliocchetti, M. Gielow, F. De Vigili, G. Conti, R. De Amicis, „A Personal Mobility Assistant based on Ambient Intelligence to Promote Sustainable Travel Choices," in *Procedia Computer Science*, vol. 5, S. 892-899, 2011.

[4] S. Bauer, „Das vernetzte Fahrzeug – Herausforderungen für die IT", in *Informatik Spektrum*, vol. 34, no. 1, S. 38-41, 2011.

[5] S. Murphy, A. Nafaa und J. Serafinski, „Advanced service delivery to the Connected Car," in IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2013), S. 147-153, 2013.

[6] A. Ranta, „Grammatical Framework: Programming with Multilingual Grammars", CSLI Publications, Stanford, CA, US, 2011.

[7] M. Heinrichs, R. Hoffmann, F. Reuter, „Mobiles Internet – Auswirkung auf Geschäftsmodelle und Wertkette der Automobilindustrie, am Beispiel MINI Connected," in Zukünftige Entwicklungen in der Mobilität – Betriebswirtschaftliche und technische Aspekte, Springer Heidelberg Dordrecht London New York, 2012, Track 3, Kap. 4, S. 611-628.

[8] N.-E. El Faouzi, H. Leung, A. Kurian, „Data fusion in intelligent transportation systems: Progress and challenges – A survey", in *Information Fusion*, vol. 12, S. 4-10, 2011.

[9] SIMPLI-CITY Konsortium: Target Market Sector Descriptor, D2.2, Public Deliverable, www.simpli-city.eu/documents.

[10] SIMPLI-CITY Konsortium: Requirements Analysis, D2.3, Public Deliverable, www.simpli-city.eu/documents.

[11] N. E. Boudette and D. Wakabayashi, "Google, Apple Forge Auto Ties," The Wall Street Journal, 29.12.2013.

[12] M. Janssen, Y. Charalabidis, A. Zuiderwijk, „Benefits, Adoption Barriers and Myths of Open Data and Open Government," in *Information Systems Management*, vol. 29, no. 4, S. 258-268, 2012.

[13] H.-L. Truong, S. Dustdar, „On Evaluating and Publishing Data Concerns for Data as a Service," in 5th IEEE Asia-Pacific Services Computing Conference (APSCC 2010), S. 363-370, 2010.

[14] C. Aggarwal, N. Ashish, A. Sheth, „The Internet of Things: A Survey from the Data-Centric Perspective," in Managing and Mining Sensor Data, Springer New York Heidelberg Dordrecht London, S. 383-428, 2013.

[18] Alle in diesem Beitrag genannten Webseiten wurden zuletzt am 08.05.2014 aufgerufen.

[15] The OSGi Alliance, „OSGi Service Platform Release 4, Version 4.3 Core Specification,” http://www.osgi.org/download/r4v43/osgi.core-4.3.0.pdf, 2011.

[16] D. Burgstahler, U. Lampe, N. Richerzhagen, R. Steinmetz, “Push vs. Pull: An Energy Perspective,” in 6th IEEE International Conference on Service Oriented Computing and Applications (SOCA 2013), S. 190-193, 2013.

[17] H.-L. Truong, S. Dustdar, “A survey on context-aware web service systems”, in *International Journal of Web Information Systems*, vol. 5, no. 1, S. 5-31, 2009.

[18] Oberlandesgericht Hamm, “Az. III-5 RBs 11/13”, 2013.