Establishing Personalized IVI Features on Distributed Open Source Webinos Middleware Using Low-cost Devices

Krishna Bangalore, Daniel Krefft, Uwe Baumgarten
Informatik
Technische Universität München
München, Germany
Email:{krishna.bangalore, krefft, baumgaru}@in.tum.de

***Abstract*— An observable trend in the automotive area leads to a growing demand for personalized in-vehicle infotainment (IVI) systems, but mostly they are closed, therefore integrating custom made apps were not possible with existing IVI-systems. So we need to extend the existing closed IVI-system with an Open source based system supporting custom made apps e.g., with diagnostic and location features, independent of the car manufacturers and the used bus-system. This paper presents a solution of a web app using low-cost devices with an Open source web-based webinos middleware. To evaluate the functionality and feasibility of the system with respect to in-car diagnostics data and location features, we present a prototype webinos *Vehicle Hub* application that runs on a HTML5 web browser which showcases the implementation of diagnostics data in dashboard view on top of the webinos middleware. Additionally for the developers to enhance the given open system we provide a *Vehicle Testbed* to test the APIs and necessary drivers. The application runs on three different device types – IVI-system, PC and smartphone/tablet. Users control the view of the dashboard and the data that they want to view with drag and drop on their PCs and dynamic streaming data with gauges on the IVI-system and smartphone/tablet.**

Keywords – ***Middleware; In-Vehicle Infotainment System; Browser; HTML5; OBD-II; Raspberry Pi; Distributed Systems; Distributed applications.***

I. INTRODUCTION

IVI-systems are following a trend in enabling in-car browser-based runtime environments [2]. Supporting IVI-systems with an embedded web browser that appears as a combined interface for cloud services and personalized features could be a starting point. Therefore, it would be important to see the capability for executing web applications or JavaScript respectively in a sufficient manner [4]. Accessing vehicle data from the Controller Area Network (CAN) bus is propriety based and is not easily available [1]. It would be important to build an open and cost effective system to access required engine data from the vehicle, following the safety and security regulations that can be used with any car. The webinos middleware provides a solution for open and web-based communication for heterogeneous devices in a distributed manner [16]. The Internet of Things Application Programming Interface (IOT API) [14] provided by webinos allows us to build drivers to connect devices like On board Diagnostics (OBD-II) [21] to be used as sensors. Based on webinos technology, the webinos *Vehicle Hub* app provides a solution for showing the OBD-II vehicle parameter values that can be viewed on the graphs and gauges, which are integrated with the webinos dashboard as an interface for managing devices and to register services in the user's personal zone or services that the user's friends provide [3]. The webinos *Vehicle Hub* app provides 18 OBD-II parameters where we can choose the parameters and set the intervals to see the values accordingly.

In summary, this paper makes the following contributions:

- We present webinos as a middleware for permitting vehicle data from the car to be viewed on a browser locally or remotely.
- We evaluate the functionality and feasibility of such a middleware approach by webinos *Vehicle Hub* application as an automotive use case.
- We additionally provide webinos *Vehicle Testbed* page for testing the APIs and necessary drivers.

The following Section 2 presents webinos personal zone concept. Section 3 shows related work about different ways of working with cars. Section 4 shows the webinos approach for connecting the in-car head unit with other devices required to build a personalized IVI-system. Sections 5 and 6 outlines the implementation and evaluation of webinos approach with a common use case. Section 7 concludes with an outlook on future work.

II. *WEBINOS* PERSONAL ZONE CONCEPT

The core webinos architecture is based on state of the art widget and HTML5 web runtime environment. The critical innovation of webinos is to place an embedded server on the devices, and place all extended APIs, policies and packaging logic behind the server [15]. By tying the

 ISBN: 978-1-61208-348-3

functionality to a server, rather than binding it to the traditional runtime, these services become addressable by other devices, not just the device the browser is running on. As shown in the Figure 1, the personal zone concepts within webinos are built up from internet agents Personal Zone Hub (PZH) and device agents Personal Zone Proxy (PZP). The way these agents communicate and identify each other, is at the heart of webinos mechanics.

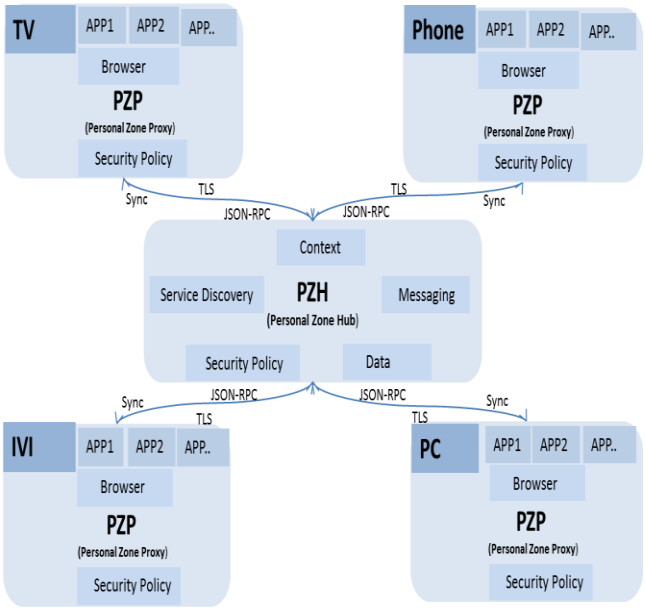

Figure 1: Personal Zone concept introduced by the *webinos* middleware

The aim of webinos is to provide a secure platform to connect heterogeneous devices like TV, IVI-system, PC and Smartphone for a multi-user and for any Operating system that supports web browsers in a web-enabled federated framework (means domains can exchange messages). The JavaScript Object Notation-Remote Procedure Call (JSON-RPC) [26] is used to get remote JavaScript and Transport Layer Security (TLS) [27] mutually authenticates the connection and gives the overlay network security and attestation. The secure session allows for transport of messages and synchronization. Apps run on top of webinos, when a webinos API is invoked, the invoked methods and its parameters are sent over a web socket, where the listening PZP checks them and returns a callback with a result locally. When it comes to remote communication the PZP forwards the request to the PZH, which finds the correct PZP that is connected to it and forwards the request. If PZP belongs to another PZH then it is forwarded under the security and policy control checks.

III. DIFFERENT WAYS OF WORKING WITH CARS

The market for personalized solutions, especially in the automotive area, is set to explode over the next few years with AutolinQ opening doors for future of mobility inside the car [17].

The following opportunities are offered by the upcoming trends:

• **Vehicle IVI-System:** A complete web based technology stack is provided to implement in-vehicle entertainment, navigation and real time driving consoles. It also provides an environment for third party application development.

• **In car communications:** A secure local communication stack allows phones, tablets, satellite navigation systems and other devices to interact seamlessly with the vehicle's entertainment and telematics systems.

• **Remote sensing:** Secure interoperable remoting protocols allow these same capabilities to be accessed by trusted remote third parties, thereby enabling remote vehicle and driver diagnostic scenarios.

A. *What is so special about webinos vehicle?*

Since there has been a huge trend for HTML5-based applications [6] Sonnenberg, presents an approach for embedding a web application server into a native application, running on a portable device [9]. Similar to that, Open source webinos middleware provides a practical solution for in-car application development and porting on to new devices. Like Chrome OS [11], Firefox OS [18] and Tizen [19] it is HTML5 based, and indeed largely compatible with these technologies. It differs in that, it is not tied to a specific application ecosystem, and comes with a suite of vehicle specific additions, which speeds up automotive application development.

A strong emphasis has been placed on the webinos security model. This is important because vehicle informatics subsystems are extremely sensitive, and grant access to highly sensitive data. Security plus interoperability is the key here. The webinos protocols are unusual in that the same mechanism that allows device interaction over the cloud can be reused for local, in-vehicle networks. For real world deployment where internet in-car is unreliable this is essential. In practice, this means that "permissions permitting" any phone or tablet in car can securely and interoperably interact with the core infotainment systems of the car. Imagine pushing the location of the destination directly from a tablet to the in-vehicle navigation or even using this exact same technology to push locations from a remote desktop [28].

Out of all the use cases that webinos supports, remote analytics and sensing are the most interesting and disruptive aspects. Fleet management, real-time logistics, remote vehicle diagnostics (automatically alerts issues similar to mbrace2) [7] and more recently, behavioral driver monitoring are all existing and in some cases quite mature technologies. The webinos technology stack is interesting in this context because it can support all of these use cases, by using entirely commoditized and Open source stacks, which not only break apart existing locked in systems, but do so in a way that grants explicit control of sensitive data to the end user.

 ISBN: 978-1-61208-348-3

IV. THE *WEBINOS* DOMAIN FOR VEHICLE

The Personal zone concept that webinos provides, the PZP and PZH are built on top of node.js [5]. Node.js is based on Google's V8 JavaScript engine, all the devices (Personal Zone Proxies) belonging to the same zone support and expose a set of standard APIs for accessing services such as device features IOT [14], Geolocation [13], Device Orientation [12], networking with other devices and cloud services.

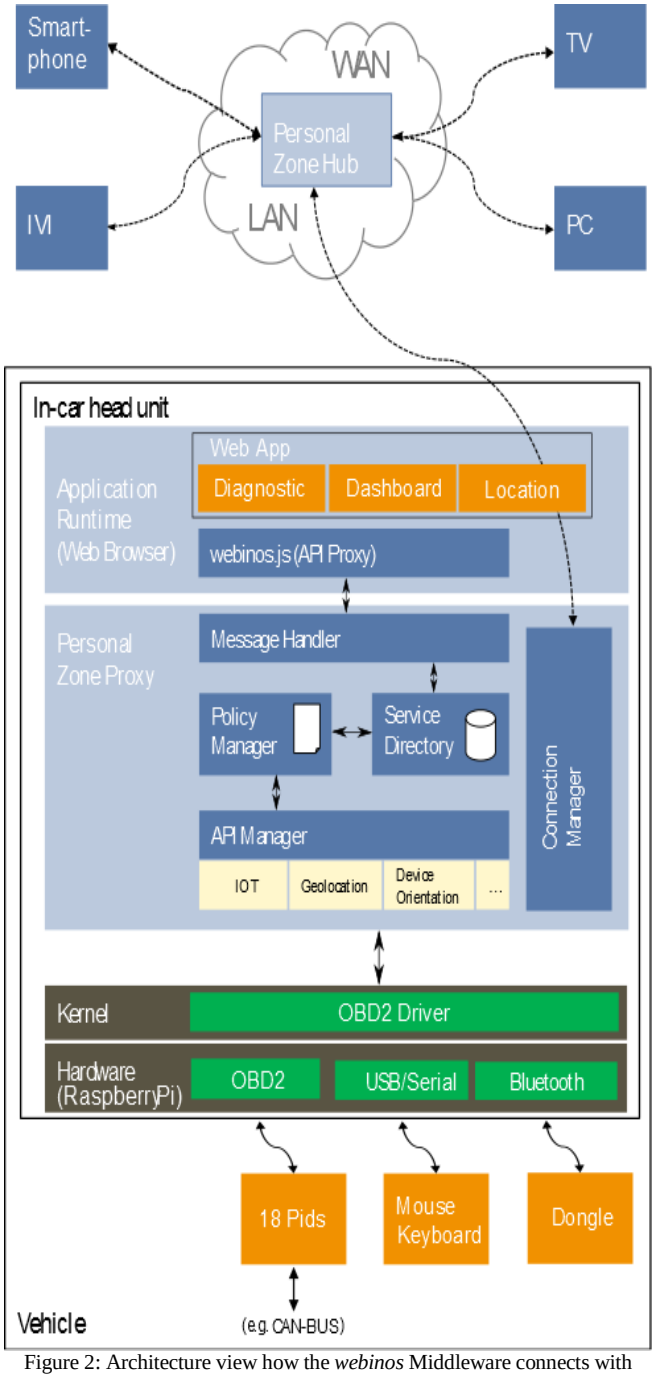

Figure 2: Architecture view how the *webinos* Middleware connects with OBD-II in a distributed automotive environment, similar to architecture [16]

As shown in Figure 2, the architecture describes the local interaction. The PZH is the center point of the personal zone between PZH and PZPs and the PZPs in the same zone if they try to share the resources. The PZPs can be more than one type of device, for example in-car devices, PCs, smartphones and so on. The interactions can take place between PZHs if the personal zones try to communicate with each other PZH.

Applications run on top of the webinos stack. PZH runs the server resident webinos applications using node.js, unpacking and performing security checks on packed widgets, authenticates users to set up trusted sessions, stores the policy files, routes messages, processes synchronization protocol messages so that the PZPs can synchronize the data from the devices [15]. The architecture describes the working of webinos middleware for the vehicle, before an API can be used by an application it has to query the Service Discovery for the available APIs for that particular PZP. Once the API is discovered it has to bind to the discovered API. In case of remotely available features the message is passed through Connection Manager, the requested feature is routed through PZH. The Policy manager checks for requests from the local or remote devices and grants access based on the policy settings set by the user.

The vehicle prototype uses Raspberry Pi [20] as hardware and the webinos IOT API that supports OBD-II driver to retrieve data from the streaming OBD-II sensor. Look for Section 5 for implementation notes.

V. IMPLEMENTATION NOTES

The interoperable specifications for the device APIs, the security model and the remoting interoperability layer have all been made available under royalty free terms. The software is highly flexible and can be deployed on several operating systems and in several configurations including Pandaboard [8], Android and Raspberry Pi.

Our current prototype deployment scenario for vehicle environment includes the following listed components attached to it (see Figure 3):

- Raspberry Pi with 5V Power-supply and a SD-Card. The Raspberry Pi does not have any internal storage, the SD-card is used to store the image of Linux version Raspbian wheezy [20]. It uses SD-card for booting and for storage. We recommend 16 GB.
- Bluetooth Dongle.
- OBD-II (Bluetooth) [21].
- Surf stick or Wi-Fi (mobile hotspot).
- Compact/mini PC display or TFT-Screen (for a closer automotive touch, we recommend to use

 ISBN: 978-1-61208-348-3

suitable car TFT with 8" to 10" display size and 16:9 ratio) [22].
- DVI-to-HDMI cable required for attaching the screen to the Raspberry Pi.
- Compact/mini PC keyboard and mouse or a combination of keyboard and mouse (pad) [23].

See our website [28] for available documentation. The source code is available under Apache 2.0 license terms.

The Bluetooth OBD-II connector is connected to the Raspberry Pi using the Bluetooth dongle. Since Raspberry Pi has two USB ports in our implementation we have connected the Bluetooth dongle and Wi-Fi stick to the keyboard which has USB hub. For the Vehicle environment we would propose to use a mini PC keyboard and mini PC mouse or a combination of keyboard and mouse (pad) [23].

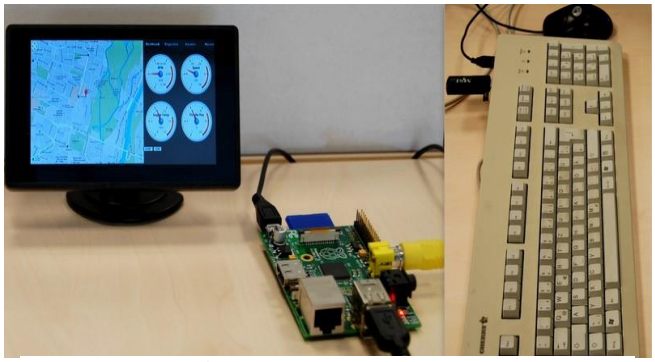

Figure 3: Prototype of Modularized webinos, running on a Raspberry Pi Hardware and showcasing IVI-system view connected to an OBD simulator

Figure 4: ELM 327 OBDII device

The ELM 327 OBD-II device (shown in Figure 4) standard specifies the type of diagnostic connector and its pinout, the electrical signaling protocols available, and the messaging format [21]. It also provides a list of vehicle parameters to monitor along with how to encode the data for each. There is a pin in the connector that provides power for the scan tool from the vehicle battery. The parameters defined in the OBD-II driver, that the IOT API uses looks for the streaming parameter messages from the OBD-II. The Geolocation and Device Orientation API offer the relevant data.

VI. EXAMPLE SCENARIO

A. *webinos Vehicle Testbed*

The webinos platform provides a *Vehicle Testbed* for testing its APIs that runs on a HTML5 browser to display test results from the OBD-II. In this particular example we connected the OBD-II to the webinos *Vehicle Testbed*, we could either test in a virgin PZP mode (not enrolled to the PZH) or enroll it to the PZH. We chose the latter and by doing this we register the browser to a particular OpenID [10] that the user wants.

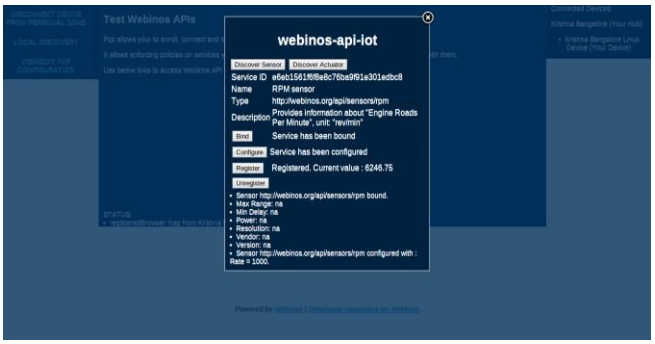

Figure 5: webinos Testbed shows RPM value

Figure 5 shows the working of webinos-api-iot, we show an example scenario of Revolutions per minute (RPM) value being streamed. We have to follow certain steps to retrieve the OBD-II value. First, find IOT API service through the service discovery. Second, bind to the found and selected service. Third, call the IOT API method to display the result. We can choose the register button as shown in the Figure 5 to connect to a listener to retrieve the RPM sensor streaming every few seconds.

B. *webinos Vehicle Hub app*

The webinos *Vehicle Hub* app shows OBD-II vehicle parameter values that can be viewed on the graphs and gauges as shown in the Figure 6 that depicts the IVI-system view and Figure 7 depicts the PC view, both use RGraph [25], which is integrated with the webinos dashboard as an interface for managing devices and to register services in the user's personal zone or services that the user's friends provided. The dashboard provides all the OBD-II parameters where we can choose the parameters and set the intervals to see the values accordingly. For demonstrating the webinos *Vehicle Hub* app we chose 5 useful parameters e.g., Engine RPM, Vehicle Speed, Throttle position, Engine Temperature and Fuel Rail Pressure that matches the webinos specifications and are useful diagnostics data.

The OBD-II drivers that are written for the webinos IOT API are registered within user's personal zone and allow the application to listen for values from OBD-II as sensors data. The application uses web technologies such as HTML5, JavaScript and CSS. For e.g., JSPlumb library [24] is used for the drag & drop feature.

 ISBN: 978-1-61208-348-3

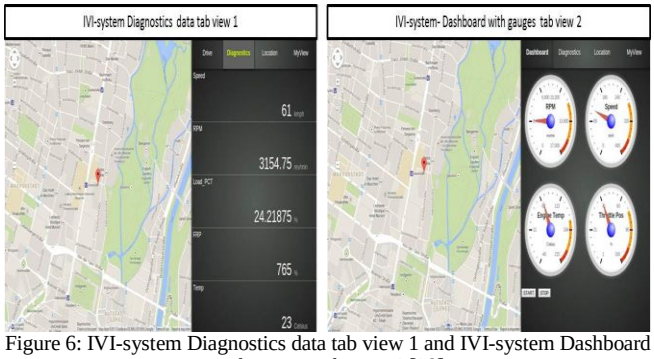

Figure 6: IVI-system Diagnostics data tab view 1 and IVI-system Dashboard with gauges tab view 2 [28]

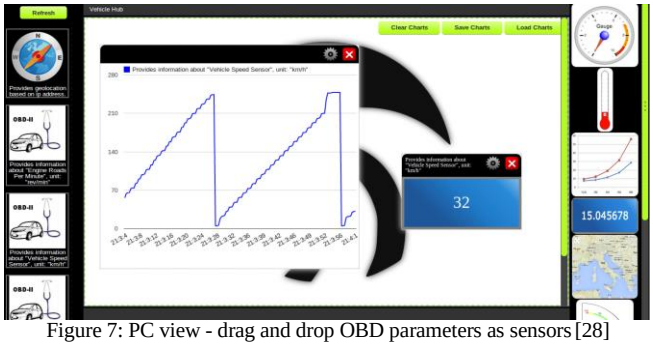
Figure 7: PC view - drag and drop OBD parameters as sensors [28]

Vehicle Hub Features:

- In the PC view we can drag and drop the OBD-II parameters on the graphs and gauges, on the right side as shown in the Figure 7 graphs and gauges are present that are to be dragged and dropped to the middle of the window screen. Each OBD-II parameter act as sensors as seen on the left side of the Figure 7 and needs to be dragged and dropped on the graph window pane to view the results.
- Shows Live streaming data both on the PC view as well on the IVI-system.
- Historical view can record and play the historical data on IVI-system, collected logs of the data from the OBD-II that could be used to show as reports, we can show them on a graph as trip data and some vehicle specific data.
- Diagnostics data shows the 18 OBD-II supported parameter values that are useful for the mechanics to diagnose the vehicle. By connecting to webinos, mechanics can view the vehicle data from elsewhere and diagnose the vehicle before it's brought to the garage.
- Insurance data collection – It was learnt from one of our workshops that the insurance companies install and maintain devices to get the vehicle reports. By using the current app and by further enhancing the webinos Vehicle Hub app we could create reports that will be useful for the insurance companies. This proposal from our workshops is now an ongoing work and is in its requirements phase.

C. *Running webinos Vehicle Hub*

Steps to run webinos *Vehicle Hub* app using OBD device with modularized webinos codebase on Linux distributions:

1. The demo works on the latest webinos version. It can be executed using firefox/Chrome browser. To make it run we need to clone the hub-webinosVehicle repository from the github [34] inside the *web_root* folder of the *webinos-pzp* or copy the content of the *hub-webinosVehicle* folder into the *web_root* folder in the *webinos-pzp*.

2. Place the *webinos-api-iot* API [30], *webinos-api-deviceOrientation* API [31] and *webinos-api-geolocation* API [32] in the *webinos-pzp* [29] folder. After that, in the *webinos-api-iot/node modules* path, clone the OBD-II drivers to get it run.

After, installing the required APIs and drivers do an *npm install* and change settings in *config.json* in *webinos-iot-driver-obd2* [33] to set the connector parameter for the Vehicle to *OBD* or if using the simulator then to *obdsim*.

Starting OBD-II simulator - The demo Figure 3 runs with an OBD-II simulator. To install the OBD-II simulator on to the Linux/Raspbian machine follow the instructions as described below.

Installing on the terminal

- *apt-cache search obd*
- *sudo apt-get install obdgpslogger*

To run with simulator

- *obdsim or obdsim –o*

Note: Change the */dev/pts/(Port Number)* in *config.json* in *webinos-iot-driver-obd2*

When connecting with a car to retrieve real time car values - Connect OBD-II (Bluetooth/Serial) into the OBD slot and rest similar to Figure 3, it runs on /dev/USB0 for serial and for Bluetooth, change the parameters of the OBD *params* settings present in the *config.json* in the *webinos-iot-driver-obd2* folder to retrieve the data.

VII. CONCLUSIONS AND FUTURE WORK

The presented work strongly focuses on providing custom apps e.g., diagnostic and location features with low-cost devices based on distributed Open source *webinos* middleware, connecting to the vehicle environment across heterogeneous devices.

This approach allows various in-car infotainment concepts. Firstly, it allows the execution of web applications that have access to the vehicle data via the introduced IOT API and OBD-II driver to support it and this implementation, can be tested on a *webinos* testbed. Secondly, the *webinos* platform

provides the components to build and communicate with the vehicle system. The implemented webinos *Vehicle Hub* highlights that the *webinos* middleware is capable and applicable to aggregate data seamlessly across heterogeneous devices, with the help of webinos dashboard. The user can control the OBD parameters that the user wishes to see.

However, the application outlines items for future work. The devices mentioned in the paper were used as a prototype to showcase the usage of *webinos* middleware. We could use a smartphone with Android operating system and OBD-II to stream the data by connecting via Bluetooth interface (which is in development) to show the application running with an automatic user interface adaption without distracting the car driver. Instead of OBD-II it would be interesting to use different devices like vehicle Black box that match the webinos specification with similar parameters. We can build innovative applications like *Insurance apps* and *Traffic apps* to support the open and web world of communication that the webinos middleware presents.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union's Seventh Framework Program (FP7-ICT-2009-5, Objective 1.2) under grant agreement number 257103 (webinos project). We thank all our project partners within the webinos consortium towards the contribution and realization of this work.

REFERENCES

[1] http://www.dgtech.com/images/primer.pdf [retrieved: May, 2014].

[2] http://www.we-conect.com/cms/media/uploads/events/31/dokumente/QNX__Why_Automakers_(Should)_Care_about_HTML5.pdf [retrieved: May, 2014].

[3] http://dev.webinos.org/deliverables/wp3/Deliverable35/wiki$t3-5$Deliverable_Specifications_Personal_Zone_Security.html [retrieved: May, 2014].

[4] S. Isenberg, M. Goebl, and U. Baumgarten. Is theWeb Ready for In-Car Infotainment? A Framework for Browser Performance Tests Suited for Embedded Vehicle Hardware. In 2012 14th IEEE International Symposium on Web Systems Evolution (WSE). IEEE, 2012.

[5] http://nodejs.org/ [retrieved: May, 2014].

[6] G. Lawton. Moving the OS to the Web. Computer, 41(3):16{19, Mar. 2008.

[7] Mercedes-Benz, mbrace. http://www.mbusa.com/mercedes/mbrace#!layout=/mbrace/remote_access&waypoint=mbrace-remote_access [retrieved: May, 2014].

[8] Pandaboard, Pandaboard Reference. http://pandaboard.org/content/resources/references [retrieved: May, 2014].

[9] J. Sonnenberg. A distributed in-vehicle service architecture using dynamically created web Services. In IEEE International Symposium on Consumer Electronics (ISCE 2010), pages 1-5. IEEE, June 2010.

[10] OpenID, http://openid.net/ [retrieved: May, 2014].

[11] Chrome OS,https://www.google.com/intl/en/chrome/browser/ [retrieved: May, 2014].

[12] W3C, DeviceOrientation Event Specification. http://dev.w3.org/geo/api/spec-source-orientation.html [retrieved: May, 2014].

[13] W3C, Geolocation API Specification. http://www.w3.org/TR/geolocation-API/ [retrieved: May, 2014].

[14] webinos, Specifications, 2014. http://dev.webinos.org/specifications/api/sensors.html [retrieved: May, 2014].

[15] webinos, Architecture, 2014. http://dev.webinos.org/deliverables/wp3/Deliverable31/wiki$wp3-1$Webinos_key_architectural_components.html [retrieved: May, 2014].

[16] Isenberg, Simon and Bangalore, Krishna and Goebl, Matthias and Haberl, Wolfgang and Baumgarten, Uwe. Towards a Personalized and Distributed In-car Infotainment Experience Using the Open and Web-based Webinos Middleware, Multi-Device '2012 [retrieved: May, 2014].

[17] Continental Corporation. AutolinQ, http://www.continental-corporation.com/www/pressportal_us_en/themes/press_releases/3_automotive_group/pr_2009_06_02_en.html [retrieved: May, 2014].

[18] http://www.mozilla.org/en-US/firefox/os/ [retrieved: May, 2014].

[19] Tizen IVI Architecture, http://events.linuxfoundation.org/images/stories/pdf/lceu2012_haitzler.pdf [retrieved: May, 2014].

[20] Raspberry Pi, www.raspberrypi.org [retrieved: May, 2014].

[21] OBD-II, http://www.obdii.com/, http://en.wikipedia.org/wiki/On-board_diagnostics [retrieved: May, 2014].

[22] TFT-Screen, http://www.cartft.com/catalog/il/1213 [retrieved: May, 2014].

[23] http://www.logitech.com/de-de/product/wireless-touch-keyboard-k400r [retrieved: May, 2014].

[24] JSplumb, http://jsplumbtoolkit.com/demo/home/jquery.html [retrieved: May, 2014].

[25] RGraph, http://www.rgraph.net/ [retrieved: May, 2014].

[26] JSON-RPC, http://www.jsonrpc.org/specification [retrieved: May, 2014].

[27] TLS, http://www.techsoup.org/support/articles-and-how-tos/introduction-to-transport-layer-security [retrieved: May, 2014].

[28] https://developer.webinos.org/vehicle-hub [retrieved: May, 2014].

[29] https://github.com/webinos/webinos-pzp [retrieved: May, 2014].

[30] https://github.com/webinos/webinos-api-iot [retrieved: May, 2014].

[31] https://github.com/webinos/webinos-api-deviceOrientation [retrieved: May, 2014].

[32] https://github.com/webinos/webinos-api-geolocation [retrieved: May, 2014].

[33] https://github.com/webinos/webinos-iot-driver-obd2 [retrieved: May, 2014].

[34] https://github.com/webinos/hub-webinosVehicle [retrieved: May, 2014].

 ISBN: 978-1-61208-348-3