Enabling Rich Web Applications for In-Vehicle Infotainment

Simon Isenberg*, Wolfgang Haberl*, Matthias Goebl*, Maximilian Michel† and Uwe Baumgarten‡
*BMW Forschung und Technik GmbH, Munich, Germany
Email: [simon.isenberg, wolfgang.wh.haberl, matthias.goebl]@bmw.de
†BMW AG, Munich, Germany
Email: maximilian.michel@bmw.de
‡Technische Universität München, Munich, Germany
Email: baumgaru@in.tum.de

***Abstract*—Web applications have been widely used on PCs, smartphones and tablets. They are now on the verge to be used for in-vehicle infotainment (IVI)-systems, as well. In order to enable a new breed of Web applications, access to vehicle data in a standardized way is required. In this paper we present application programming interfaces (APIs) for accessing vehicle bus data from a standard Web browser. The APIs have been developed as part of the webinos project. We evaluate the feasibility of our approach with a prototype implementation of a browser-based Point-of-Interest manager and a trip computer using these APIs.**

***Keywords*-Web applications; in-vehicle infotainment; API; browser; vehicle data**

I. Introduction

Web browsers have made the transition from pure document viewers to runtime environments for applications [1]. Standardization activities around HTML 5 and commercial products such as Chrome OS and Firefox OS highlight this paradigm shift. The Web is going to connect not only desktop and laptop PCs, but also mobile devices, home entertainment systems, and potentially in-vehicle infotainment (IVI)-systems. Research activities [2] and announcements by QNX [3] to feature Web technology in their upcoming QNX Car 2 platform or by Intel and Samsung about TIZEN IVI [4] emphasize the effort towards a browser based runtime environment on IVI-systems.

The use of Web technology for application development has advantages compared to native solutions [5] [6] [7]. In respect to IVI-systems the ease of development, deployment, and portability of Web applications makes the Web browser the preferred candidate for a runtime environment. The current landscape of IVI-systems is highly fragmented with different OEM specific solutions. The fragmentation hinders attraction of third-party developers for native apps and increases development costs for IVI-systems. Another result of the fragmentation is the unavailability of applications and services on IVI-systems to customize the user experience.

Contrariwise, the highly standardized and less fragmented Web technology provides the desired range of applications, which is backed by a large developer base. Using the browser as a runtime environment on IVI-systems can help to lower development costs, while providing a wide range of applications at the same time.

However, standardized APIs for Web applications to interact with the vehicle systems are missing. Such APIs must take the automotive constraints including limited hardware resources and sandboxing of infotainment applications from the general vehicle system into account.

In summary, this paper makes the following contributions:

- We present two JavaScript APIs for interacting with the vehicle, the *Vehicle API* for retrieving vehicle specific data from bus systems and the *Navigation API* for interacting with the on-board navigation system.
- We have implemented the APIs on top of the webinos platform[1] and evaluated the feasibility of our APIs with two prototype applications.

In the following Section II we discuss the design of the Vehicle API and Navigation API as part of browser based application runtime webinos. Section III outlines our prototype implementation. We emphasize the feasibility of our approach by showing a browser based trip computer and Point-of-Interest (POI) manager in Section IV. The paper concludes with a summary of our findings and an outlook on future work.

II. API Design

While designing both APIs, we obeyed the following principles:

1) Avoid exposing data duplicates which are already available from other standardized APIs such as the Geolocation API[2].
2) Follow general design principles for Web APIs which are encouraged by W3C and use an asynchronous model to retrieve dynamic vehicle data.
3) Group vehicle properties to be easily used in Web applications.

As the Geolocation API provides information about the position including latitude, longitude, altitude, speed and heading of a device, we do not incorporate these data

[1]http://www.webinos.org
[2]http://www.w3.org/TR/geolocation-API/

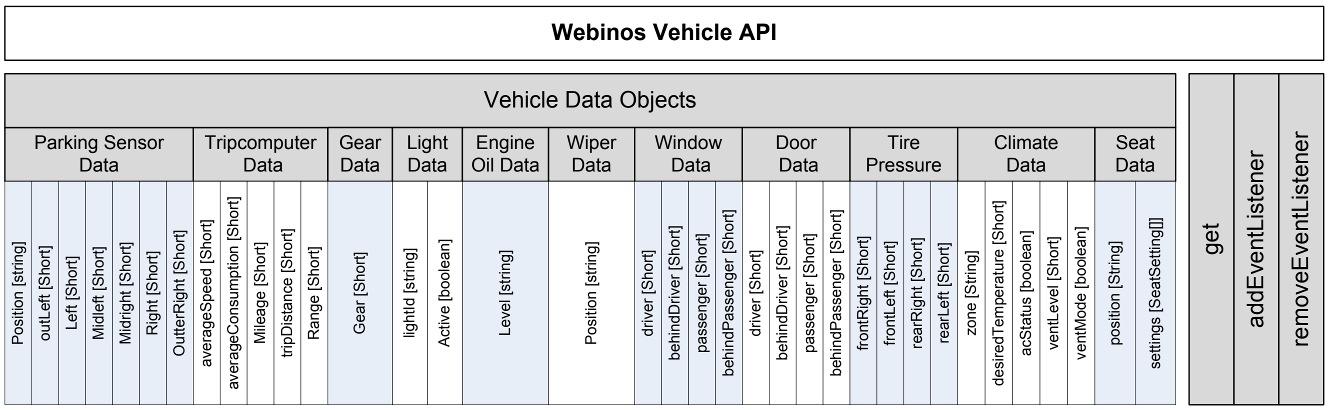

Figure 1. Dynamic Vehicle Data Provided by the webinos Vehicle API

properties into our Vehicle API. The same applies for lateral and longitudinal acceleration, which is provided by Device Motion Events of the Device Orientation API[3]. Instead an implementation of the existing API specifications for the automotive platform is used.

A. *API for retrieving vehicle data*

Because of the frequent use of an event-based design in JavaScript APIs as seen in the Device Orientation API our Vehicle API is influenced by W3C's DOM Level 3 Events model[4]. We provide vehicle data which change frequently in an event-based manner. An app developer can register listeners on specific vehicle properties or can request vehicle data in a non-recurring way. Static information about the vehicle such as brand, model, etc. is provided as separate attributes of the vehicle object. Figure 1 illustrates, which data properties are exposed by our Vehicle API and can be retrieved using the *addEventListener()* or *get()* functions. The vehicle data is passed back to the defined callback handler. We defined eleven different data objects for retrieving vehicle data. Web app developers gain access to (1) park sensors at the front and the rear of the vehicle, (2) trip computer data such as average fuel consumption, (3) the current gear, the state of (4) lights, (5) windows, (6) doors, (7) wipers and (8) tires. Additionally, information about (9) air conditioning settings, (10) seat settings and (11) engine oil level are exposed, as well.

Listing 1 illustrates how to add a callback listener on trip computer data. Whenever one of the five trip computer data properties (average speed, average consumption, mileage, trip distance or range) changes, the registered callback is being triggered with the updated trip computer data.

The vehicle data properties are grouped based on two reasons: On the one hand, they are grouped in a way that they can be easily used by applications. On the other hand, groups are designed to minimize the access frequency to the different bus systems and to reduce the overhead for pushing the vehicle data from the different buses into the browser engine.

```
webinos.vehicle.addEventListener("tripcomputer", tcHandler
    );
function tcHandler(e){
 alert(tc.averageSpeed);
}
```

Listing 1. Adding an Event Listener on Trip Computer Data

The full specification of the Vehicle API is available at http://dev.webinos.org/specifications/new/vehicle.html.

B. *API for interacting with the on-board navigation system*

Besides the ability to read vehicle data from the bus system, we provide mechanisms to interact with the on-board navigation system using the Navigation API. The separation between navigation features and vehicle data access is motivated by the fact that the navigation features are not exclusively provided by vehicles, but could also be available on smartphones.

The Navigation API exposes four different functions for interacting with the satellite navigation system and supports

- to query the navigation system for POIs in customizable area,
- to hand over a POI or address to the navigation system and request guidance to it,
- to cancel guidance to a given POI,
- and to retrieve the state of the navigation system (active, inactive).

When an application passes a POI to the navigation system, a callback handler can be registered. The handler supports mechanism to notify the application, if the destination has been successfully passed to the navigation system (*onRequest()*), the guidance to the destination has been cancelled (*onCancel()*) or the destination has been reached

[3]http://www.w3.org/TR/screen-orientation/
[4]http://www.w3.org/TR/DOM-Level-3-Events/

((*onCancel()*). Listing 2 illustrates how to pass a POI to the navigation system and make use of the callback handler. The full specification of the Navigation API is available at http://dev.webinos.org/specifications/new/navigation.html.

```
var navigationHandler = {
 onRequest:  function(id, poi){
  console.log('Guidance␣set␣to' + poi.name);
 },
 onReach: function(id, poi){
  console.log(poi.name + '␣reached.');
 },
 onCancel: function(id, poi)
               console.log('Guidance␣to␣' + poi.name + '␣
                    is␣cancelled.');
       }
var destination = {name:"BMW␣Group␣Research␣and␣Technology
     ", address:{street: "Hanauer␣Strasse", streetNumber:
     "46", postalCode: "80992", city: "Muenchen", country:
      "DE"};
webinos.navigation.requestGuidance(destination, false,
     navigationHandler);
```

Listing 2. Handing over a POI to the navigation system

III. Prototype Implementation

In order to provide vehicle bus data and interact with the on-board navigation system we are accessing two different vehicle buses. First we have a module for accessing the Media Oriented Systems Transport (MOST)-Bus and second for accessing the Controller Area Network (CAN)-Bus.

The architecture to access the vehicle data from the Web browser is depicted in Figure 2. The module to access the car data is decoupled from the browser engine. This approach is the integral part of the webinos middleware as described in detail in [8]. It allows us to provide access to vehicle data from Web applications which are executed on different devices, such as a connected smartphone.

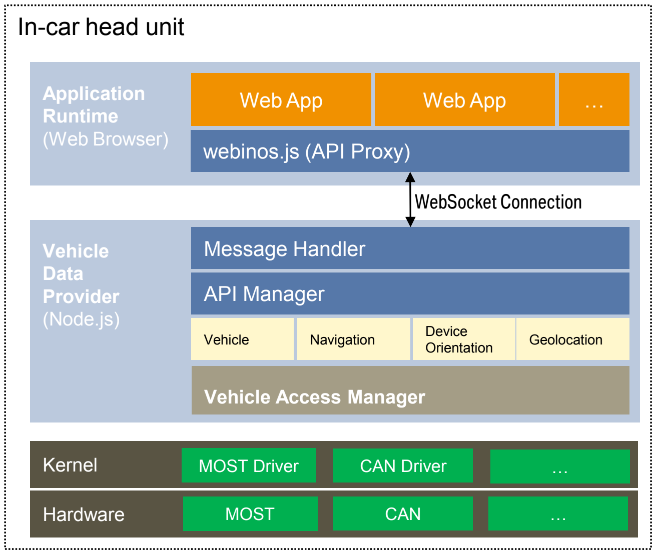

Figure 2. Architecture for accessing bus data

When a Web application is launched, the browser injects a JavaScript library to provide access to the Vehicle, Navigation, Geolocation and Device Orientation API. The library establishes a connection to the *Vehicle Data Provider* using a WebSocket connection. The Vehicle Data Provider is implemented as a Node.js[5] application. Node.js is a platform for building network applications on top of Google Chrome's JavaScript script runtime. Our Vehicle Data Provider contains a native Node.js add-on called *Vehicle Access Manager* for accessing the MOST- and CAN-Bus.

From a developer perspective the injected library acts as a regular JavaScript API. The library handles requests to the supported APIs. It wraps the requests into JSON-RPC request and passes it to the Vehicle Data Provider running on Node.js. The Vehicle Data Provider extracts the request and passes the request to the internal *API manager*, which is wrapped around the Node.js add-on for accessing the vehicle buses.

Our Vehicle Access Manager parses the different bus messages converts the relevant messages into the various vehicle data or position and device motion event objects. The same applies to the Navigation API. The created JavaScript objects are than passed back to the browser inside a JSON-RPC response. The injected JavaScript library extracts the response message and triggers the registered callback functions of the Web application with the provided data object. Our prototype system runs on top of an optimized version of Ubuntu 11.10 and supports ARM and Intel x86 architectures.

IV. Demo Applications

In order to evaluate the feasibility of our concept we created two browser applications, which make use of the newly introduced APIs. Both apps have an optimized user interface for in-car usage and can be controlled using the BMW iDrive controller[6]. Due to webinos' cross-platform approach, the applications can be executed on multiple devices as displayed in figures 3 and 4.

Figure 3 depicts a browser based trip-computer. The application visualizes the data which can be retrieved form Vehicle, Geolocation and Device Orientation API as a general trip computer. The user can customize different views and specify which vehicle properties, shall be displayed. As shown in the figure, the app maybe executed on in-car head units, tablets or smartphones.

Figure 4 shows the POI manager called *webinos travel*. The application enables the user to create POIs on any webinos enabled device, such as a smartphone, pc or tablet. The information about the different POIs is synced between the different CE-devices using the webinos middleware. On the IVI-system, the user can browse the created POIs and

[5]http://nodejs.org
[6]http://www.bmw.com/com/en/insights/technology/technology_guide/articles/controller.html

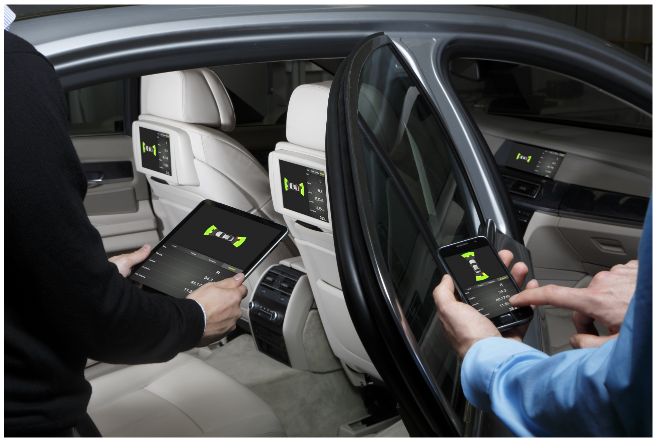
Figure 3. webinos trip computer (Courtesy of BMW AG)

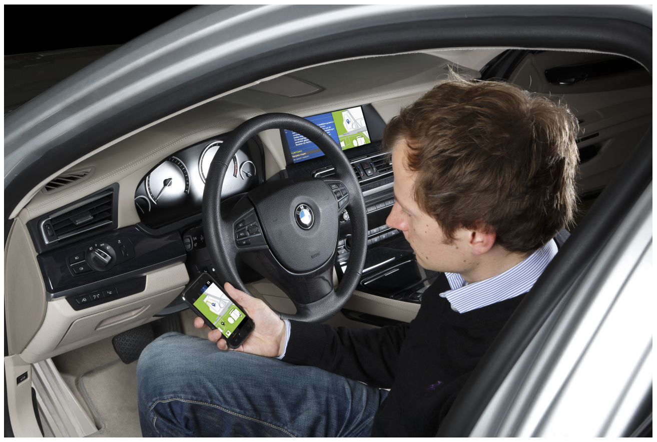
Figure 4. webinos travel (Courtesy of BMW AG)

passes the POI to the on-board navigation system using the Navigation API.

V. Summary & Future Work

The paper at hand puts a strong focus on accessing different vehicle busses and exposing the vehicle specific data in a generic way for Web applications. The Vehicle API and Navigation API provide the basis for richer Web based IVI-applications. The API provides unique device data, which is usually not available to Web applications. In order to commercially use these APIs, fine-grained access control mechanisms have to be integrated into the solution. Such a fine-grained access control is also necessary before allowing Web apps to set certain vehicle properties.

Additional items for future work in respect to Web and Automotive would cover ways to automatically adapt the content of the Web applications to the available input controls inside the vehicle and adjust the graphical user interface to minimize driver distraction.

Acknowledgment

The research leading to these results has received funding from the European Unions Seventh Framework Program (FP7-ICT-2009-5, Objective 1.2) under grant agreement number 257103 (webinos project).

References

[1] A. Taivalsaari and T. Mikkonen, “Objects in the Cloud May Be Closer Than They Appear,” in *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*. IEEE, Sep. 2011, pp. 59–64.

[2] 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.

[3] A. Gryc, “Why Automakers (Should) Care about HTML5,” 2011. [Online]. Available: http://support7.qnx.com/download/download/22989/qnx_auto_html5.pdf/download/download/22989/qnx_auto_html5.pdf

[4] M. Ylinen, “Tizen IVI Architecture,” 2012. [Online]. Available: http://download.tizen.org/misc/media/conference2012/wednesday/bayview/2012-05-09-0945-1025-tizen_ivi_architecture.pdf

[5] S. Adler, “WebOS,” *netWorker*, vol. 9, no. 4, pp. 18–26, Dec. 2005.

[6] G. Lawton, “Moving the OS to the Web,” *Computer*, vol. 41, no. 3, pp. 16–19, Mar. 2008.

[7] A. Taivalsaari and T. Mikkonen, “The Web as an Application Platform: The Saga Continues,” in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, Aug. 2011, pp. 170–174.

[8] C. Fuhrhop, J. Lyle, and S. Faily, “The webinos project,” in *Proceedings of the 21st international conference companion on World Wide Web - WWW ’12 Companion*. New York, New York, USA: ACM Press, Apr. 2012, p. 259. [Online]. Available: http://dl.acm.org/citation.cfm?id=2187980.2188024