2010 IEEE Intelligent Vehicles Symposium
University of California, San Diego, CA, USA
June 21-24, 2010

A SOA-based Middleware Concept for In-vehicle Service Discovery and Device Integration

Michael Eichhorn*, Martin Pfannenstein*, Daniel Muhra**, Eckehard Steinbach*
Institute for Media Technology
Technische Universität München
Email: *{firstname.lastname}@tum.de, **muhra@in.tum.de

Abstract— **We present a novel middleware approach for in-vehicle service discovery and device integration that is based on the concept of Service-oriented Architectures (SOA). In order to be able to identify a suitable SOA system, we define a series of criteria the SOA platform should fulfill. These criteria take into account the specific requirements of the automotive domain. Based on these criteria we compare nine different SOA standards and show why the *Device Profile for Web Services* (DPWS) standard is most suitable as the middleware of an in-car infotainment and communication system. Furthermore, we present a prototypical implementation of an in-vehicle network based on the DPWS middleware. Hence, we enable the integration of nomadic devices by providing their functionalities within the network and consequently including their graphical output into the in-vehicle Human Machine Interface (HMI). Due to the system design it is also possible to use nomadic devices as an additional display to present the HMI. Finally, an in-vehicle controller is included into our SOA and therefore available to all other devices within our system.**

I. Introduction

Modern cars feature a multitude of driver assistance, entertainment and comfort functions. The more sophisticated and expensive a car is, the more functions are typically included. These functions can be quite complex and usually go far beyond simple sensors and actuators. Examples for sophisticated functions are a night vision system, a lane departure warning system or an advanced parking assistant based on multiple cameras. Up to now, the involved Eletronic Control Units (ECUs), controllers and displays are interconnected directly or connected via a dedicated vehicle bus to the systems they are supposed to communicate with. For example, a controller which is used to interact with the Human Machine Interface (HMI), is usually connected via the Controller Area Network (CAN) to the central driver information screen where the HMI is also displayed. This system then listens, among others, to the corresponding commands coming from the controller on that specific bus. In addition, there are several vehicle busses deployed which are used simultaneously in modern cars to cover different domains like infotainment, power train communication, safety applications and so on. Examples for such vehicular busses are the Media Oriented Streaming Bus [1] which is mostly applied in the infotainment domain and FlexRay [2] for X-by-Wire applications. In the field of inter-ECU (electronic control unit) connects, e.g. small sensors, actuators or engine control, Local Interconnect Network (LIN) [3] and CAN [4] networks are widely deployed. These two standards are also widely used in the automation control sector. In addition to these two wired networks, there are also wireless technologies applied, generally referred to as WPANs (Wireless Private Area Networks). WPANs are covered by [5] and can be found in technologies like Bluetooth [6] and Zigbee [7]. Another approach which is used in automation control is Z-Wave [8]. As for cost reasons, wireless interconnections between ECUs have not yet found their way into the car, wireless technologies are not regarded here. The increasing number of ECUs and newly introduced services in a car, cause the need for expensive and complex gateways between the vehicular busses in order to support the exchange of data across domain boundaries. Therefore,

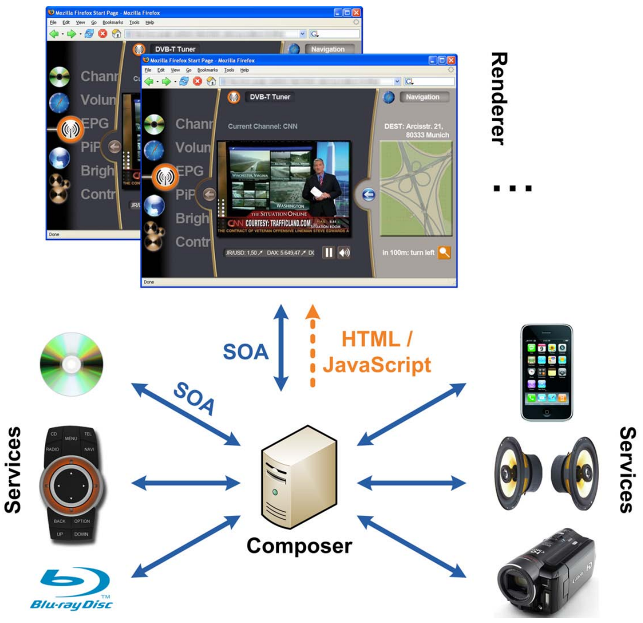

Fig. 1: Example devices that are represented and act as services. The graphical output is composed from various content sources. The device announcement is supposed to be handled by the Service-oriented Architecture.

a common communication infrastructure is desirable. With the increasing cost pressure vehicle manufacturers are faced with today, there is a trend towards in-car deployment of Ethernet / IP, which are widely used and proven standards in the consumer electronics (CE) domain. An Ethernet-based in-vehicle network was, for instance, presented in [9]. We therefore take IP as a basis for our proposed system. A distinction between IPv4 and IPv6 is not considered in our system as it works with both addressing standards. Currently, we rely on an IPv4 addressing scheme. Despite the fact that also in a future in-vehicle network, ECUs will be separated into different domains due to security issues, we consider in this paper a scenario where all ECUs can interact with each other. Furthermore, such a future network or at least a network's subsegment should be open to nomadic devices like laptops or smartphones. In order to use all of the features built in today's nomadic devices across the network, we rely on a Service-oriented Architecture [10] (SOA). Figure 1 shows an overview of the scenario which was also introduced in [11].

In Figure 1, multiple devices like a controller, a mobile phone or a DVD player are integrated into the system. Not only should the devices be accessible within the network via the Service-oriented Architecture, but certain devices might also provide a graphical user interface which should then be integrated into the user HMI which is shown at the very top of Figure 1.

This paper is organized as follows. In Section II, we discuss related work from the automotive domain. In Section III, our middleware concept is introduced. Furthermore, an overview of the service setup is given. The system design of our SOA approach is introduced in Section IV. In Section V, we define the criteria we use to select a SOA platform which meets our requirements and the decision for one specific middleware system is explained. Section VI then describes our prototypical realization. Our conclusions and an outlook on future work are presented in Section VII.

II. Related Work

In the automotive domain, one can find several recent approaches towards the realization of more flexible on-board infotainment systems. One example is Continental's AutoLinQ [12] platform. This Google Android [13] based system can be extended by applications and therefore be customized to the driver's preference. However, neither the integration of services is intended, nor does AutoLinQ provide a middleware for network-wide use of services or the applications installed. Many vehicle manufacturers partly open their systems in order to integrate consumer electronics (CE) devices. One popular example which is featured in nearly all major cars as an optional accessory is a connector for Apple's iPod. Having the iPod connected to the car, the name of the current audio track is displayed on one of the on-board displays. It also allows the driver to control the iPod via the car's entertainment system or even by pressing some buttons directly at the steering wheel. One first approach that extends the built-in functionalities in a car is the "smart drive kit" [14] for the Apple iPhone which is available for Daimler's Smart series. With this kit it is possible to expand the on-board computer of a compact car as this usually does not come with advanced features of today's premium cars. In a future version, the iPhone is supposed to communicate with a WiFi camera that is mounted behind the windshield to detect traffic signs.

III. Overview

Figure 2 gives an overview of the proposed service architecture. In Section IV, the boxes (1) and (2) are described in more detail. The entity numbered (1) represents our man-

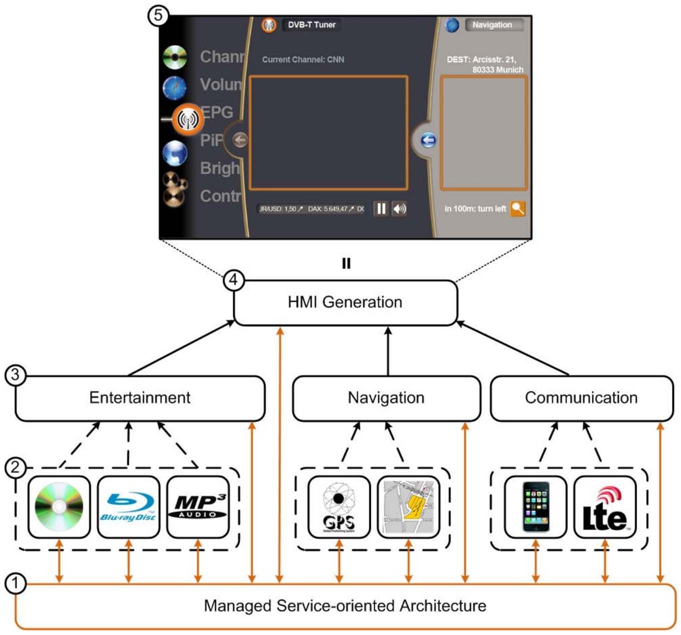

Fig. 2: Service setup with HMI output: (1) SOA platform; (2) devices, i.e., services; (3) UI generation of the single services condensed to logical menu items; (4) HMI generation, i.e., the user menu; (5) graphical output.

aged Service-oriented Architecture. All services announce themselves at this entity when entering the network or at the start-up phase of the whole system. Services can be the devices numbered (2). E.g., the GPS receiver (4th device from the left) as well as a map server (5th device from the left) offer their services by announcing them to the SOA. The menu assembly for the submenu "Navigation" is also handled by a service (3). This service is not provided by a dedicated hardware device, but realized as a software component, i.e., a service within the SOA. The different submenus are then composed to finally generate the HMI by the "HMI Generation" block (4). Even this entity is realized as a service in our system. The output (5) is then presented at an arbitrary display device which invokes the "HMI Generation"-service (4).

IV. System Design of the SOA

In SOA-based approaches, the system environment is modeled by the use of components. Basically two main types can be found: services and clients. Those two are not distinct and in fact it is often the case that a component acts as a service on the one hand, but on the other hand requires services for that purpose and acts thereby as a client. For the most naive scenario, every component would know about every other one in the system and would also be allowed to use them. Within a car, this is not sufficient since there have to be privileged services, e.g., a steering wheel which is not allowed to be replaced, as well as exclusive services, not accessible by everyone, for instance due to safety reasons. To ensure this distinction, we propose several techniques for service discovery and authentication. In the following, we will explain why almost every component is a service and point out the need for a central service registry.

A. *Service provider*

As mentioned before, access right handling is one of the key issues for the design of services. To manage this at a central component would be possible, but very inflexible since every component would have to be known in advance. Therefore, one would have to update this core component or perform a reconfiguration of all service providers every time the configuration needs to be extended. In general, most devices will provide more than one single service which share the same access rights. It is also meaningful to delegate the access control to the device itself by using a key-based authentication method. By that, it is possible to grant newly joined clients access to the device. Therefore, it is necessary to equip the client with a unique key. From a development perspective, this would only require an additional service for authentication on the (service) device.

To enforce higher security, we propose not only controlled service access, but also controlled service discovery. This is not only useful to prevent misuse and Denial of Service (DoS) attacks, but also exclusively supplies clients with services of their interest. For that, we introduce a discovery proxy (DP) as a trusted third party between the service and the client. Consequently, the proxy now handles the authentication for the client and provides it with service information. For that purpose the keys of the client must be accessible and again those can be provided using a service. Compared to other security approaches the use of keys is a simple solution. We therefore want to address this issue in our future work by introducing certificates and encryption. We aim at an open and scalable security which can be handled by embedded devices as well as state-of-the-art electronic devices.

B. *Discovery Proxy*

As the name implies, the discovery proxy mainly manages the distribution of service information. It does not actively look for new services or devices, only in case of testing availability, a polling mechanism is provided, but it rather serves as a central registry, where every device signs up. The desired behavior is not to promote those services to all others, but make explicit distinctions. Basically, the proxy deals with two different scenarios:

1) Client signs on: In principle, the client wants to get information on all services it can use and it is allowed to use. Since it is possible that new services sign on later, it is meaningful to let the proxy notify the client about them. In both cases, the client has to provide authentication keys for the services it wants to invoke. At this moment it acts as a service provider for the proxy. In Figure 3 a message pattern diagram during the registration of a client at the DP is shown. The client starts with a `HELLO` message. After that, the DP is asking for the appropriate keys by sending `getKeys()` and in turn receives keys from the client with `sendKeys(k...)`. Now, the DP can send the specific keys to devices that match the security scope of the client by `validateKey(k)`. Their response indicates if the key is accepted or not (`accept/reject`) and therefore whether the client should be granted access to the device. At last, as soon as a device signs off, the proxy has to notify all clients that were aware of it. Figure 3 shows the `BYE` message from the device to the DP followed by the notify message `deviceGone(A)` which is sent to the client.

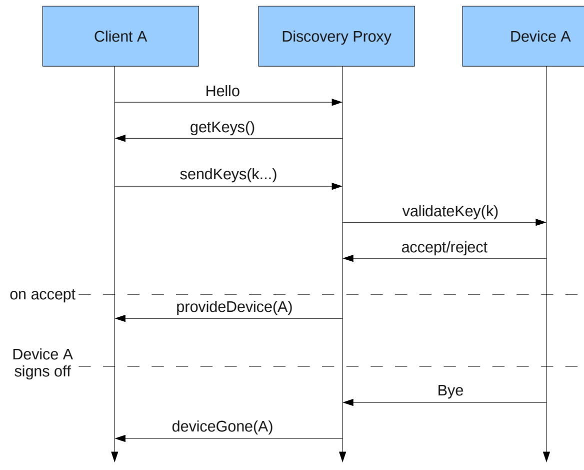

Fig. 3: General client registration scheme.

2) Service provider (device) signs on: Compared to the first scenario, this one is rather short. A diagram of the messages exchanged can be seen in Figure 4. First of all, the device checks whether it also serves as a client and adapts its procedure if necessary. After registration by the message `HELLO`, the DP subscribes to the device to receive the notification of its status. This happens during the service call `subscribeToStatus()`. Next, a check on whether the device is ready to serve is performed. Due to a lack of services which are mandatory in order for the device to be fully operable, it is possible that the services on the device are not available at the beginning. In such a case, the proxy would have to wait until the device notifies its availability. This is implied by the status change from ”not ready” to ”ready” in Figure 4. Until then, it will be treated as a client only. As soon as the device becomes available, the proxy will try to announce its state, respectively, its information to requesting clients according to the described procedure by `provideDevice(A)`.

C. *Advantages*

The main advantage of the proposed system is its flexibility. Devices, respectively their introduced services, are

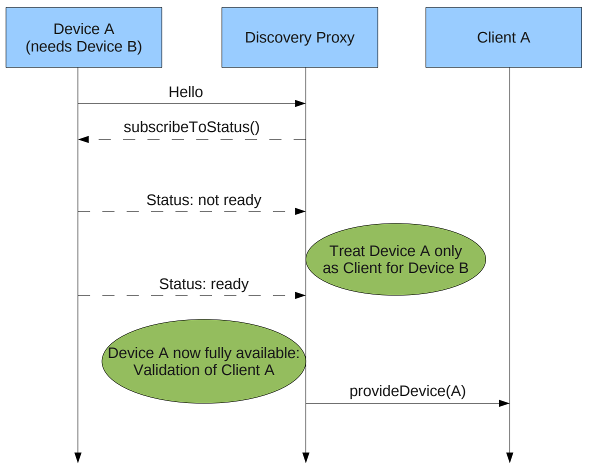

Fig. 4: General device registration scheme. Dashed lines represent an asynchronous message exchange, solid lines represent a synchronous message exchange.

available network-wide due to the SOA and the corresponding announcements. Therefore, they can be used by all other services. Input and output devices can be integrated into the SOA and be used flexibly rather than being connected to the corresponding system in a hardwired fashion. The built-in output devices in a car like displays can now be used by, e.g., external input devices and vice versa. On the one hand, this introduces a huge benefit towards the development of new features offered by a car, but on the other hand, it also challenges the system design due to security issues. A further advantage of the discussed system is its high scalability because of the modular approach. A newly introduced device has to be compatible to the system in terms of announcement and communication patterns, but is then fully integrated into the system. No changes to the DPWS stack are necessary to integrate existing DPWS devices. Accessory equipment can be integrated flexibly depending on the car line and the user's preferred configuration. Also new features realized by software which rely on the devices integrated in the network can be enabled by updating the system.

V. Realization of the SOA

A. Selection Criteria

We define selection criteria that can be used to identify an appropriate SOA system that matches our requirements. An overview of the SOA platforms considered in this work is given in Table I. The rating ranges from "++" "fully supported", "+" "supported with need for modifications", "–" "implementable with high development effort" to "––" "not possible". A "0" stands for "neutral" respectively "no information provided".

Low resource requirements: Within the automotive domain, it is important to have low resource consumption, meaning that the applied software should have low requirements on the hardware it is supposed to be deployed on. In turn, there are also constraints on the hardware like low energy consumption, low base price and reliability. Therefore, the support of the SOA regarding embedded systems is preferred.

Peer-to-Peer (P2P) concept: With respect to failsafe operation of a middleware which is indispensable within a vehicular environment, a peer-to-peer operation of the message exchange scheme is preferred. The announcement and authentication has to be performed against a central entity (which can also be setup in a failsafe manner) but the service calls should be done directly from one service to another rather than involving a central entity.

Eventing: Subscription and event notification are important and elegant messaging concepts that cut message traffic dramatically in comparison to polling based approaches. Since most services within a car act event based, its support is crucial.

Plug&Play: Devices should be integrated during runtime of the system. Therefore, a seamless reconfiguration of the SOA without pausing the regular operation is mandatory. Device discovery and announcements should work without any user interaction, if possible.

Device support: Devices can introduce multiple services which should then be available system-wide. The different services on the device should be handled separately and without any cut-back on the service functionality during simultaneous operation.

State: Part of the message flow between devices will not only depend on the participants but also on their internal state. This should also be considered to avoid unnecessary message overhead.

Filtering: With the aspect of plug&play in mind, security should not be compromised. Devices should only know about other devices related to their own functionality or ones that are in the same security context. Also filtering with regard to access rights should be considered.

Network traffic: The traffic in the in-vehicle network should be kept as low as possible, due to the costs of energy and the used network infrastructure. Furthermore, sufficient bandwidth for critical services like airbag sensors must be reserved at all time.

Platform issues: Our approach should support as many devices as possible. Therefore, a wide platform coverage as well as support for multiple programming languages is desired. It is necessary to offer a platform independent solution to increase the general acceptance.

Security: Security is very important in the in-vehicle environment because of the need to protect the intellectual property of the car manufacturer, i.e., the HMI content. Furthermore, a car is open to malicious attacks that can be performed from the inside or outside. A SOA should provide mechanisms to avoid these threats.

Code generation: Code generation is an important aspect for the developers of applications who want to release services for our architecture. Also the market acceptance would benefit from lower development costs.

Flexibility: In principle, our platform is open to any types

	DPWS	WS	REST	CORBA	Java RMI	UPnP	Jini	HAVi	OSGi
Low resource requirements	++	0	0	−−	−−	+	−−	−	+
Peer-to-Peer (P2P) concept	++	++	−−	++	+	++	−−	−−	−−
Eventing	++	−	−−	−−	+	++	+	+	+
Plug&Play	++	−	0	+	+	+	+	+	−
Device support	++	−	−	+	+	+	+	+	+
State	++	++	−	++	+	++	+	+	+
Filtering	0	0	−−	0	+	0	+	+	+
Network traffic	++	+	−	0	++	−	++	+	++
Platform issues	++	++	0	0	++	0	−	+	−
Security	++	++	+	0	++	−	+	+	+
Code generation	++	++	0	+	0	0	0	0	0
Flexibility	++	++	−	++	−	−	+	+	+
Enc. of graphical information	0	0	0	0	0	0	0	0	0

TABLE I: Comparision of different SOA standards. ("++" "fully supported", "+" "supported with need for modifications", "–" "implementable with high development effort" "−−" "not possible", "0" "neutral / no information provided").

of devices. Therefore, we need a SOA which comes with a high flexibility concerning the supported communication patterns and management features.

Encoding of graphical information: Certain devices respectively services will be used to present graphical information (like watching DVD, placing a call...). Their graphical presentation will depend on the actual devices being used. Therefore, there will be a need to exchange graphical information on a template basis.

Real-time behavior is a current issue among middleware systems. Nevertheless, this aspect is not considered within this comparison. As can be seen in [15] and [16], a real-time extension is hard to manage and comes along with several limitations. First of all, real-time always relies on the lower network capabilities which have to meet certain time constraints in order for the middleware to be real-time enabled. As we take IP as a sublayer and neglect lower layers, real-time is not within the focus of this comparison.

B. Comparison

Since most of our requirements are addressed by several frameworks, APIs and standards, we give an overview of selected technologies in Table I. We further explain in detail, why we have selected the Device Profile for Web Service [17] (DPWS) to be the basis of our system.

Java RMI, Jini and OSGi are all possible approaches for middleware-based solutions. In general, they all have their own advantages, e.g., OSGi [18] brings a very intuitive service description realized by Java interfaces. An advantage of OSGi is the possibility to deploy applications [19], which in our system is not of importance. For security reasons we do not want to dynamically deploy applications. The physical representation of one application should be read-only. Thus, we do not consider further runtime environment based methods. Nevertheless, a combination of OSGi and DPWS is also considerable in order to benefit from Web Services as well as from OSGi ([20]). Unfortunately, even small and resource-constrained devices like single sensors would need to have the Java virtual machine implemented, which would lead to an unacceptable overhead.

HAVi's [21] focus lies on home domain applications and it is still restricted to IEEE 1394 networks. Although it offers plug&play capabilities, the limitation to the IEEE 1394 network disqualifies it for our purposes.

CORBA [22] is a very promising approach for middleware solutions, as it offers independency regarding the applied platform, hardware and programming language. Unfortunately, it never met higher market acceptance and the highly abstract definition used opposes intuitive approaches.

Very popular among today's internet services is REST-based [23] communication. It only defines 6 different actions (GET, POST, PUT, DELETE, HEAD, OPTIONS), therefore the behavior stays pretty predictable. Another important aspect of this technology is the use of stateless clients. Hence, any state information must be transmitted within every message. Since this dramatically increases the computational complexity as well as the message sizes, we decided against a REST implementation.

Another standard provided by the World Wide Web Consortium (W3C) [24] is Web Services [25]. This defines platform independent interfaces, which are described by WSDL (Web Services Description Language). Several additional specifications exist which allow Web Services to make use of advanced features like eventing, discovery and addressing. Still, there is no sufficient concept for device integration or plug and play functionality.

Universal Plug and Play (UPnP) [26] comes close to our requirements. It's a simple middleware focusing on small networks, usually in the CE domain. Mechanisms for discovery, plug&play and eventing are implemented. However, on the one hand, it lacks of support for IPv6 [27], on the other hand, serious security issues remain. Since we require a management mechanism for device discovery and authentication, we moved towards a standard that would provide us with raised possibilities regarding those particular requirements.

DPWS (Device Profile for Web Services) [28] is a standard which was designed for the use of Web Services on embedded and even deeply embedded devices [29]. It was first published in May 2004 and since June 2009, it is an official OASIS standard [28]. The core features include (managed) dynamic discovery and eventing. As it takes Web Services as a basis, it guarantees platform and programming language independency. Furthermore, it uses device models to bundle services and organize them into logical units. For these reasons, we decided to use DPWS as the backbone of our system. As DPWS meets our requirements closely, we only had to extend the managed discovery to our needs. For that purpose, we could build on top of a discovery proxy, which is also part of the standard.

C. Implementation

There are two well-known implementations of the DPWS standard, WS4D (Web Services for Devices) [30] and SOA4D (Service-Oriented Architecture for Devices) [31]. Due to the better documentation and easier integration possibilities with respect to our extensions, we decided to use SOA4D. Within this implementation, all of the core elements of the DPWS standard are included.

VI. Prototypical Realization

The DPWS middleware described in Section V was prototypically implemented into a lab demonstrator. Here, the SOA is used to make use of the system-wide availability of a controller. In Figure 5, our lab demonstrator is shown. The single components are numbered from (1) to (6). Component (1) is a WiFi router which connects all our IP-based systems. They can be hardwired or wirelessly connected like for example an Apple iPhone or an Android-based mobile phone (6) which are used to display HMI content in our scenario. Component (2) is a lightweight web server based on an ARM-platform. The whole web server architecture only consumes about 8 MB of disc space. The embedded display unit of the demonstrator consists of the components (3) and (4). Entity (3) is an Intel Atom-based PC which is connected directly to an in-vehicle display (4). As the display is driven by LVDS (low voltage differential signaling), the DVI output from our embedded PC (3) has to be converted. This is done by a dedicated box not shown here. Furthermore, we included an in-vehicle controller (5) into the system which is connected via CAN. The CAN signals are converted to IP packets by a dedicated gateway in order to make the commands from the controller available in our infrastructure.

In our scenario, the web server provides HMI content realized by web technologies as described in [11]. This content is rendered by the embedded PC (3) which acts as a client. The output is then presented on the in-vehicle display (4). We use DPWS to offer the controller's commands within the network. The client can then subscribe to this specific service and interact with the HMI. Due to the SOA, the commands can now be used by multiple clients. E.g., a laptop can wirelessly join the network, open a browser and subscribe to the controller service. The eventing mechanism from DPWS is then enabled and the browser "listens" to the controller commands. Thus, the whole HMI can be controlled by the controller but contrary to the common in-vehicle infrastructure design, the controller can also be used in another perspective. As an example, we show the integration of nomadic devices by displaying the HMI on an Apple iPhone. The controller service can also be invoked

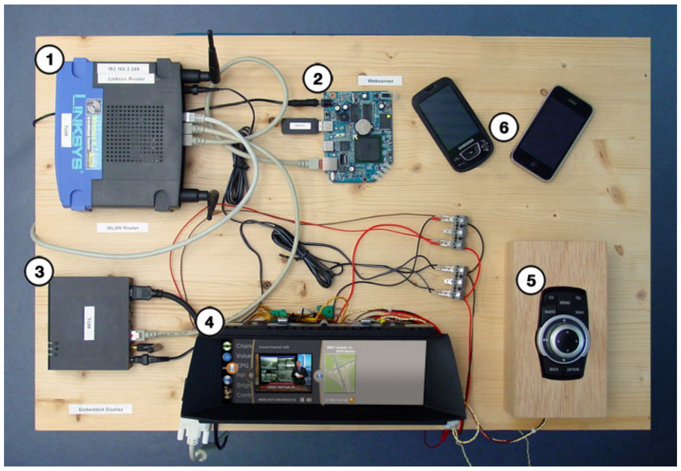

Fig. 5: Lab Demonstrator. (1) WiFi router; (2) lightweight web server; (3) embedded PC as client for the display; (4) in-vehicle display; (5) in-vehicle controller; (6) nomadic devices (smartphones).

from the iPhone. As a result, the controller can then be used to interact with the in-vehicle HMI displayed on the iPhone.

VII. Conclusion and Outlook

In this paper, we present a middleware concept for an in-vehicle network. Furthermore, a comparison of different Service-oriented Architectures was given with respect to our requirements. We decided to go with DPWS as it covers the most crucial requirements like platform independency, low resource consumption and various security aspects. Furthermore, it is aligned with the Web Services standard which guarantees a broad development community as DPWS includes the core specifications of Web Services. Our main focus is on the device integration as well as on dynamic HMI generation. The first is mainly covered by DPWS. Hence, we do not have to extend the DPWS implementation to include a management of the services and to address some of the automotive-related security aspects. We have to provide management and security services based on DPWS in order to accomplish the described features of our architecture. Every DPWS compliant device can therefore be used within our infrastructure by implementing our extensions. With respect to the scope of this paper, the HMI generation was not described. We intend to use the SOA to dynamically generate the user HMI by means of distributed services within the network. This will also be a main part of our future work. An user interface (UI) will be provided by several services within the network. The generation of the actual HMI should then be handled by dedicated services which invoke the UI content of the devices and assemble a submenu. The main HMI is then also generated by a dedicated service as shown in Figure 2. In our future work, we will concentrate on security aspects to protect the HMI content respectively the intellectual property as well as to protect the system against malicious threats from the inside as well as from the outside.

VIII. Acknowledgment

This work has been supported, in part, by the BMBF funded research project SEIS (Security in Embedded IP-based Systems) [32].

References

[1] MOST Cooperation, "Media Oriented Streaming Bus," http://www.mostcooperation.com/home/index.html, last accessed Feb. 2010.

[2] Altran Technologies, "Flexray," http://www.flexray.com/, last accessed Feb. 2010.

[3] LIN Consortium, "Local interconnect network," http://www.lin-subbus.org/, last accessed Feb. 2010.

[4] ISO 11898, *Road vehicles – Controller area network (CAN) – Part 1-5*. ISO, Geneva, Switzerland, 2003-2007.

[5] IEEE, "802.15.4 standard," http://www.ieee802.org/15/pub/TG4.html, last accessed May 2010.

[6] ——, "Bluetooth standard," http://www.ieee802.org/15/pub/TG1.html, last accessed May 2010.

[7] Zigbee Alliance, "Zigbee," http://www.zigbee.org/, last accessed May 2010.

[8] Z-wave Alliance, "Z-wave," http://www.z-wavealliance.org, last accessed May 2010.

[9] B. Mueller-Rathgeber, M. Eichhorn, and H.-U. Michel, "A unified Car-IT Communication-Architecture: Design Guidelines and prototypical Implementation," *In Proc. of the IEEE Intelligent Vehicles Symposium (IV)*, Sept. 2008.

[10] T. Erl, *"Service-oriented architecture: concepts, technology, and design"*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.

[11] M. Eichhorn, M. Pfannenstein, and E. Steinbach, "A flexible in-vehicle HMI architecture based on web technologies," in *International Workshop on Multimodal Interfaces for Automotive Applications (MIAA2010)*, Hong Kong, China, Feb. 2010.

[12] Continental Automotive GmbH, "AutoLinQ," http://www.autolinq.net/, last accessed Feb. 2010.

[13] Google Inc., "Google Android," http://code.google.com/android/, last accessed Feb. 2010.

[14] Daimler, "Smart Drive Kit for the iPhone." http://www.daimler.com/dccom/0-5-633234-1-1274927-1-0-0-0-0-0-16694-0-0-0-0-0-0-0-0.html, last accessed Feb. 2010.

[15] G. Moritz, S. Prüter, D. Timmermann, and F. Golatowski, "Real-Time Service-oriented Communication Protocols on Resource Constrained Devices," in *Proc. Vol. 3 of International Multiconference on Computer Science and Information Technology, Wisla, Poland*, October 2008, pp. 695–701.

[16] S. Prüter, G. Moritz, E. Zeeb, R. Salomon, F. Golatowski, and D. Timmermann, "Applicability of web service technologies to reach real time capabilities," in *ISORC '08: Proc. of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008, pp. 229–233.

[17] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski, "Service-oriented architectures for embedded systems using devices profile for web services," in *21st International Conference on Advanced Information Networking and Applications Workshops*, 2007.

[18] R. Hall and H. Cervantes, "An OSGi implementation and experience report," in *IEEE Consumer Communications and Networking Conference, CCNC 2004*, Las Vegas, Nevada, USA, January 2004, pp. 394–399.

[19] IST Amigo Project, "Amigo osgi programming framework," http://amigo.gforge.inria.fr/home/components/wp3/OSGi_Framework /index/index.html, last accessed May 2010.

[20] A. Bottaro, A. Gérodolle, and S. Marié, "Combining OSGi technology and Web Services to realize the plug-n-play dream in the home network," *OSGi Community Event, Munich, Germany*, 2007.

[21] R. Lea, S. Gibbs, A. Dara-Abrams, and E. Eytchison, "Networking home entertainment devices with HAVi," *Computer*, vol. 33, no. 7, pp. 35–43, 2000.

[22] S. Vinoski *et al.*, "CORBA: Integrating diverse applications within distributed heterogeneous environments," *IEEE Communications Magazine*, vol. 35, no. 2, pp. 46–55, 1997.

[23] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs." big"web services: making the right architectural decision," *International World Wide Web Conference, Proceeding of the 17th international conference on World Wide Web*, 2008.

[24] W3C, "World wide web consortium (w3c)," http://www.w3.org/, last accessed Feb. 2010.

[25] E. Newcomer and G. Lomow, *"Understanding SOA with Web Services (Independent Technology Guides)"*. Addison-Wesley Professional, 2004.

[26] Contributing Members of the UPnP Forum, "Universal Plug and Play Forum," http://www.upnp.org/, last accessed Feb. 2010.

[27] C. Li, Y. Huang, and H. Chao, "UPnP IPv4/IPv6 Bridge for Home Networking Environment," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 1651–1655, 2008.

[28] OASIS, "Devices Profile for Web Services Version 1.1," http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html, last accessed Nov. 2009.

[29] G. Moritz, S. Prüter, D. Timmermann, and F. Golatowski, "Web services on deeply embedded devices with real-time processing," in *Proc. of ETFA 2008*, September 2008, pp. 432–435.

[30] University of Rostock, "Web Services for Devices," http://www.ws4d.org/, last accessed Nov. 2009.

[31] SOA4D Forge, "Service-oriented Architecture for Devices (SOA4D)," https://forge.soa4d.org/, last accessed Feb. 2010.

[32] EENOVA, "SEIS (Security in Embedded IP-based Systems)," http://www.eenova.de/projekte/seis, last accessed Feb. 2010.