Practical Experiences on a Communication Middleware for IP-based In-Car Networks

Kay Weckemann Hyung-Taek Lim Daniel Herrscher

BMW Group Research and Technology
Hanauerstr. 46
Munich, Germany
{Kay.Weckemann, Hyung-Taek.Lim, Daniel.Herrscher}@bmw.de

ABSTRACT

Current in-car communication networks are based on automotive specific technologies like CAN, FlexRay and MOST and use proprietary communication protocols. While CAN and FlexRay come with a signal-based communication paradigm, MOST provides a more sophisticated interface based on "function blocks" to the application programmer. In the next years, we expect IP-based protocols and standard technologies like Ethernet to be deployed for more and more in-car communication tasks. As a result, we need an IP-based communication middleware that provides both signal- and function-based interaction paradigms and works for all distributed applications in vehicles. Main challenges are the large variety of embedded devices and operating systems used in a single car in terms of footprint and compute power. Furthermore, it must be possible to migrate existing interface definitions from legacy technologies to the new IP-based solution. In this paper, we propose an IP-based in-car middleware framework based on an open source solution, Apache Etch. We sketch how different, yet interoperable versions of the middleware can be used to construct a scalable system that fits to both small and large devices. Finally, we identify extensions to Etch that are necessary to qualify the solution for the use in the automotive domain.

Keywords

IP-Based In-Car Networks, Industrial Adaption of Middleware, Embedded Middleware

1. INTRODUCTION

The increasing variety of networking technologies used in automobiles today results in complex and cost-intensive E/E architectures. A modern vehicle features a complex communication infrastructure: up to seventy electronic control units (ECUs) are interlinked with up to five networking technologies that in turn are connected to each other. These technologies come with custom protocols that are not directly compatible. Therefore, a special application layer gateway is needed to interconnect them. Furthermore, innovations call for more bandwidth, which again is going to introduce additional technologies and protocols to the car. The growing complexity becomes an impediment to innovation. We propose to greatly reduce the complexity by introducing the internet protocol (IP) as the standard inter-domain networking protocol for demanding use cases in the vehicle [8].

IP was invented to facilitate the coupling of heterogeneous networking technologies - exactly the situation we face in our cars. Today, we have several CAN (Controller Area Network [6]) subnets using the CAN packet format, a MOST (Media Oriented Systems Transport [13]) subnet using MOST frames and so on [14]. It is rather straightforward to envision an all-IP in-car network that consists of several IP subnets instead, connected by an IP router. Each ECU would be assigned an IP address, and therefore could easily communicate with any other ECU in the car. But this scenario lacks realism, mainly because IP communication requires a networking technology underneath that is capable to transport an IP datagram of a certain size. Most of the current ECUs in a car are connected via CAN or LIN (Local Interconnect Network [15]), however. While it is possible to fragment an IP packet to transport it over CAN and even LIN, this is far from being efficient and reasonable - and upgrading the CAN nodes to e.g. Ethernet just to make them IP-capable would be too expensive.

Nevertheless, there are candidates for a gradual migration from automotive-specific to IP-based communication. Several ECUs handle much data: The navigation system works on a large map data base, infotainment devices can be connected to the user's mass storage (e.g. via USB), always-on cars communicate over the internet, and driver assistance cameras stream high-quality live video data. Last but not least, several ECUs work with multi-megabyte software, which has to be updated on a regular basis. These devices would benefit from a high-bandwidth connection today, and we propose this high-bandwidth connection to be realized by an IP-capable networking technology.

Using IP networking in the car features two main benefits. First, standard hardware that can transport IP is today cheaper than comparable automotive-specific networking hardware, at similar or even better bandwidth. The best

COMSWARE 2011, July 04-07, Verona, Italy
Copyright © 2012 ICST
DOI 10.4108/comsware.2011.10

candidate technology for high-bandwidth communication in cars today is Ethernet. Since 100 Mbit/s Ethernet is available in a two-wire unshielded version, it has become even more interesting [4]. Second, using IP as a standard protocol facilitates to use the whole set of standards around IP: Starting with TCP and UDP (yes, CAN and MOST come with their own transport protocols, "CAN TP" and "MOST High"), but also a variety of standard application protocols are beneficial for in-car use cases. To give just one example: Sharing a storage device using an automotive bus is a complex and error-prone task - use NFS over IP and it's done. In this scenario, it becomes clear that we also need a standard way for in-car application interaction on top of IP. On CAN, data are transferred on the basis of simple signals. The coding of these signals is done by convention in a simple, yet efficient binary form. In contrast, MOST, a technology aiming at in-car digital audio transmission introduced in the 1990s, comes with a more sophisticated middleware solution based on "function blocks". This middleware is today even used for some local interprocess communication tasks ("virtual MOST").

A future IP-based in-car network also needs to provide a standardized way for application interaction - providing at least the features of MOST and CAN, but also being future proof. At present, there is not an off-the-shelf solution that covers the embedded requirements typical for CAN, the complex interaction requirements of MOST and the future flexibility that we expect from an IP-based solution. Furthermore, to the best of the authors' knowledge, no published work lists the special software requirements for an automotive IP-based in-car network. In this paper, we present such requirements and our proposed solution – adapting an existing open source middleware to become automotive.

The remainder of this paper is organized as follows: First, we present motivating use cases and discuss the most important requirements for a communication middleware in the automotive context in section 2. In section 3, we conclude the main functional parts of an automotive middleware from these requirements. Afterwards, we give a short description of our favorite solution Apache Etch in section 4. Finally, we conclude our paper with an outlook on how we see the project evolving in chapter 5.

2. AUTOMOTIVE REQUIREMENTS FOR AN IP-BASED COMMUNICATION MIDDLEWARE

This section presents motivating use cases and important requirements for a communication middleware applied in an in-car network.

2.1 Use Cases

Since we are aiming at a general automotive middleware, we should not tailor our solution to few use cases. However, sketching important candidate use cases helps to understand the general requirements given afterwards. The first use cases anticipated for an in-car IP-network are functions that transfer much data, like sources and sinks for digital streaming, and devices that communicate with the internet.

- IP-cameras: Currently, digital cameras used for driver assistance are usually connected by an analog or digital video signal cable, and a CAN communication link to control the imager. An IP-camera would stream digital, most likely compressed video data, and would be controlled by IP-based middleware messages. Typical messages are start / stop, brightness settings, synchronization messages. The streaming data is not transferred by the middleware, but a specialized protocol (like RTP). The communication is visualized in figure 1.

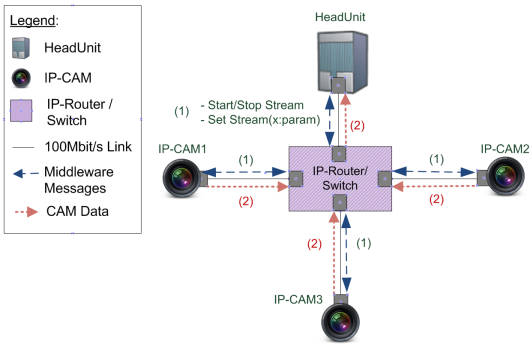

Figure 1: Visualization of the communication for the IP-camera use case.

- IP-amplifier: A typical external amplifier in a car is currently connected either by analog audio cables and a CAN control, or by MOST, used for both digital audio and control of the amplifier settings. Again, an IP-based counterpart would stream the digital audio data with a standard protocol, and use an IP-based middleware for control settings like volume, mute commands, fading and mixing control to handle different sources, and equalizer settings. This is visualized in figure 2.

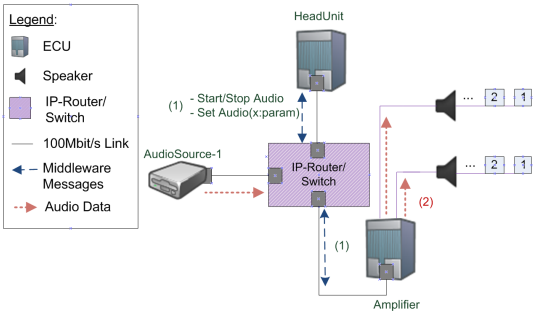

Figure 2: Visualization of the communication for the IP-amplifier use case.

- IP-radio module: An IP-based car would have a device to connect it to the internet, of course. But a car does not only send and receive internet data (e.g. via GSM) - it is also a receiver for several broadcast services like radio and TV standards, GPS, etc. It makes sense to combine several radio frontends into one ECU to minimize antenna and cabling costs. An integrated IP radio module could serve as universal receiver for multiple services, internet access, and positioning. The IP

middleware would be used to tune the receivers, communicate the current position on a regular basis, and control the internet connection (e.g. dialup). Video and audio streaming is done by other protocols.

2.2 Real-Time Properties of Ethernet

As introduced earlier, Ethernet is the best candidate technology for high-bandwidth communication in cars. Although not in the focus of this paper, but important for the listed use cases, this subsection briefly mentions recently done analysis on Ethernet real-time properties. We differentiate between the need for continuous audio and video streams and event-based control messages. Continuous streaming, as needed in the examples above, can easily be handled using adequate buffer strategies. CAN- and Flexray-like control messages require more thorough inspection. An extensive view on automotive Ethernet challenges with respect to real-time is presented in Lim et al. [12]. Based on that, in [11] the same authors present a performance evaluation of in-car switched Ethernet with several traffic types including control data. The end-to-end delay requirement of 10 ms for time-critical data can be met in their simulation using an IEEE 802.1Q priorization mechanism. Although studies are ongoing, this serves as a first evidence that automotive real-time requirements can be met in a well-defined in-car Ethernet.

2.3 Functional Requirements

The transport of control data is executed via communication middleware to give the application developer abstraction from ECU distribution. Aside there also exist specific control protocols for media transport. For instance, media streaming over the Real Time Protocol (RTP) features a dedicated control protocol named RTCP (RealTime Control Protocol) [16]. Another example can be found in the Audio Video Bridging (AVB) standard specified as IEEE 1722. AVB comes with the extension IEEE 1722.1 that defines service discovery and control of IEEE 1722 devices [9, 10]. In such cases, it does not seem reasonable to replace the standard control protocols by a communication middleware. Instead, these protocols are used as application specific control mechanisms in parallel to our middleware solution. However, the possible coexistence of other control protocols does not impose additional requirements on our solution and therefore is not discussed further in this paper.

For the middleware, the minimal subset for interaction consists of a common serialization on the representation layer and a Remote Procedure Call (RPC) functionality on the session layer [3]. Additionally, a notification functionality is required as alternative communication paradigm to map the similar functionality in MOST (MOST notifications) and CAN (signal abstraction). UDP and TCP serve as standard transport protocols, dependent on the specific application requirements.

Figure 3 presents the functional components of the in-car middleware above the TCP/IP stack. The middleware is located on top of an existing transport protocol, most likely TCP or UDP, although it is not yet clear if the respective full standard implementation can be used on automotive devices and operating systems. The middleware's interface to the application layer is given through skeleton code for

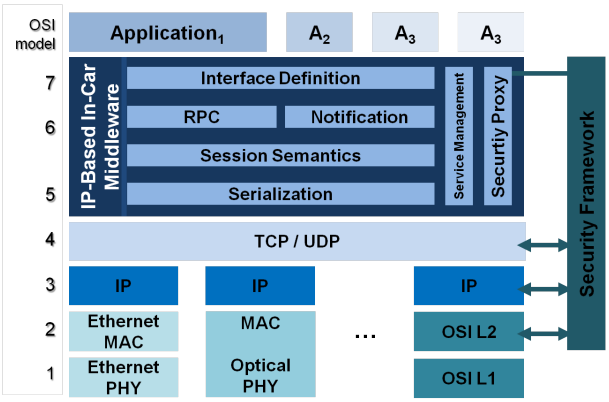

Figure 3: Overview of the IP-based middleware and its functionality within the OSI model

a chosen programming language binding as specified in the communication-relevant interface definitions.

Let's start with the Interface Definition: A middleware must provide an interface definition language (IDL) with defined syntax and semantic that allows the description of communication interfaces independent of a concrete programming language. To deliver concise definitions, an object-oriented style of the IDL is desirable. Requirements for the interface definition language are the ability

- to define the communication functionality – RPC and notifications including the respective session semantics
- to describe typical simple data types (int, float, string, and so on) as well as a selection of complex types (array, list, and so on)
- to stipulate exception handling and
- to allow for a modular design.

The middleware tool chain must include a compiler that automatically generates stub and skeleton code in selected programming language bindings. Program code in the automotive industry is typically written in C and C++, recently also in Java.

This program code must implement all software parts necessary, so that the application programmer can develop on local representatives of the remote application interface. This is what the components RPC and Notification present in figure 3. This includes the Session Semantics as well. Session semantics describe guarantees about the possible frequency of a function call execution. We distinguish between 'maybe', 'at-least-once', and 'at-most-once' execution [18]. Implementing these semantics requires timeouts and maintaining lists of message acknowledgements. In the wider sense, semantics of communication also includes the decision whether to communicate synchronously or asynchronously – the latter requiring processes to run multi-threaded [2].

The middleware must provide and implement serialization rules, so that remote applications on different platforms pro-

duce and consume defined, binding data streams. The serialization transforms a complex object on a certain platform and language binding into an ordered block of data that can be de-serialized on another platform and language binding. Serialization can be optimized on various interdependent dimensions: littleness, serialization time, and maintainability – just to mention those relevant to us. Finding the right trade-off needs thorough evaluation that cannot yet be provided in this paper.

Besides, system wide communication services are needed: The most important ones are service and security management. The number of ECUs varies from a few to 70 devices in contemporary premium cars and they are not all powered at all times. Additionally, the cautious opening of the in-car network to exchangeable parts and more dynamically through the integration of Consumer Electronics (CE) devices, demands for service discovery. The greater challenge to service management will be posed by the requirement to manage the status of a service. Sub-network operation is gaining importance for addressing energy efficiency from the E/E perspective, especially with electric vehicles ahead. While it should be possible to shut down parts of the communication network dynamically, the safe operation of the vehicle must always be guaranteed.

Security is increasingly becoming critical with an IP-based in-car network due to openness and more people already familiar with the technology. As securing the network not only affects the communication software, it is covered by a more general security framework. In this context, we refer to the research project EVITA that developed an architecture which provides modular security services [21]. These services can be adapted with regard to different security zones. A defined security interface can therefore be implemented by multiple plug-ins. The communication middleware accesses the security interfaces through a security proxy which is part of the middleware component to provide secured communication according to selected security goals [20] [21].

2.4 Non-Functional Requirements

Not every distributed application in the vehicle has the same requirements to communication services. Especially applications on powerful ECUs with high functionality work on large and complex data structures, show dynamic behavior, and therefore demand complex interaction mechanisms. The arising challenge is scalability in terms of required hardware power. In today's cars, there is a wide variety of ECUs with CPUs of different processing powers. There are needs for communication of an up-to-date 32-bit ECU with a small 16-bit or maybe even 8-bit sensor device with memory space varying in a great range. The answer to which is the smallest ECU to be considered is rather difficult: not all ECUs will be equipped with an Ethernet controller. We expect bandwidth to be the critical decision point. High bandwidth can thoroughly go along with small memory. The aforementioned camera use case is a good example for high bandwidth streaming along with a very small memory footprint. A practical consequence of the memory restriction is the fact that a standard TCP implementation is not applicable here. It is necessary to use either a reduced TCP implementation for embedded devices, or to use UDP only and provide connection-handling within the middleware or the application if needed.

2.5 Contradicting Requirements

On the one hand, we stated functional requirements to support several communication paradigms and system-wide communication services. On the other hand, we require the middleware solution to run even on small devices. The actual problems small ECUs have with complete middleware implementations are:

- The dynamic management of notification lists and service tables requires memory. Furthermore, it is hard to estimate how much memory has to be reserved for these lists at specification time.
- The transmission of large and dynamic data structures requires a dynamic memory management.
- Session-oriented interaction patterns need either TCP or a session handling on top of UDP. Both solutions require extra memory and an operating system supporting multi-threading.
- Security features (encryption, authentication) need extra memory and compute power. Secure key storage requires extra hardware.

It becomes obvious that the optimal solution to our middleware requirements cannot be run on small ECUs. However, we aim at an interoperable system that spans small and large ECUs. To solve this conflict, we propose the design of different derivates of the middleware that are optimized to the complexity of applications and power of hosting ECUs.

2.6 Designing Interoperable Middleware Derivates

Therefore, a middleware for contemporary vehicles should offer a fully featured specification for powerful devices and at the same time provide an interoperable feature subset, which supports the communication needs of small sensor devices. This fact can be referred to as 'functional scalability'. Functional requirements for automotive IP-based middleware consequently vary with application complexity and computing power respectively: The minimal subset for interaction must at least consist of a static definition of data syntax and a dynamic communication protocol. Based on the functional requirements above, this will be implemented through a defined subset of serialization rules that allow the small-footprint device to statically allocate memory beforehand. The dynamic communication protocol is a simple RPC where the application must assure the session semantic if needed. Before this idea of having several middleware derivates is concretized in section 3, a discussion of requirements is necessary that result from this concept. First, communication must be possible between any IP-based ECUs. This can lead to different behaviors of a powerful device when communicating with another powerful device compared to communicating with a restricted device. Another requirement is an inclusion demand that forces a larger specification to include all functionality of a smaller one, for the reason to comply to the typical development path over time (see figure 4). The advantage lies in the possibility to migrate

any further-developed application to a more powerful ECU without having to maintain the communication interface of the legacy code.

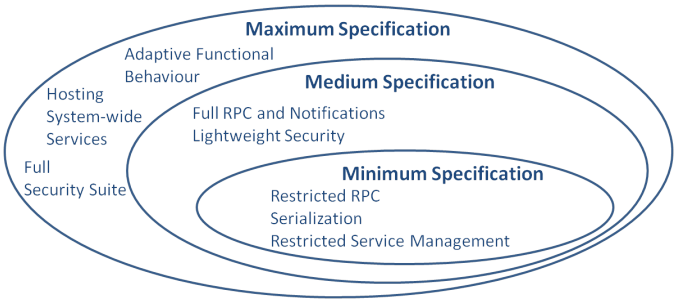

Figure 4: Schematic presentation of the inclusion demand for the functionality of different specifications

3. CONCEPTIONAL PARTS OF AUTOMOTIVE IP-BASED MIDDLEWARE

This section presents and discusses the components of different specifications of the middleware solution, trying to incorporate the identified requirements.

Our present considerations give evidence that two specifications of the middleware will satisfy the automotive requirements. For the time being, our prototype environment offers such a maximum and a minimum specification which is sufficient to answer most questions concerning the realizability. Nonetheless we add an abstract medium specification to our concept to allow for easy adaption of a future special use case. The differentiation between these three categories serves to illustrate the ideas of specification, design and interoperability.

The definition of a specification is motivated by use cases and the compute power of typical hardware platforms for the respective task.

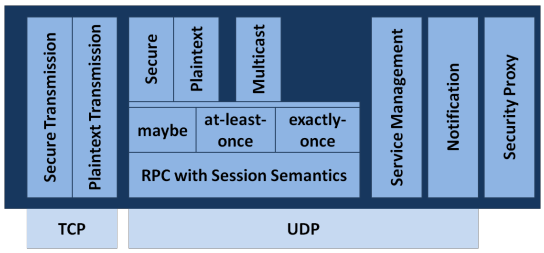

Figure 5: Functional Components of the Maximum Specification

The maximum specification offers the full spectrum of possible communication paradigms, session-semantics, and additional communication services: remote procedure calls with chosen semantic, notification service and service discovery and options to enforce security requirements.

The medium specification offers a subset of the maximum one that is optimized for less CPU and memory usage but still offering most of the communication options. This specification not designed yet, figure 6 shows optional fields in

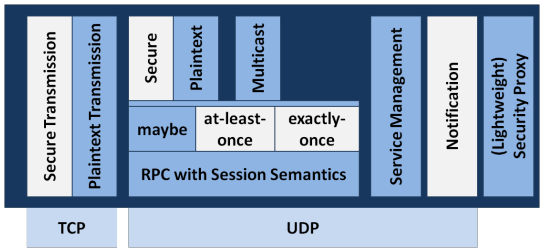

Figure 6: Functional Components of the Medium Specification

grey. It is an open issue how much flexibility the medium specification should keep. Varying the medium specification between ECUs offers design space for optimization, while defining one fixed medium specification eases software maintenance and network validation.

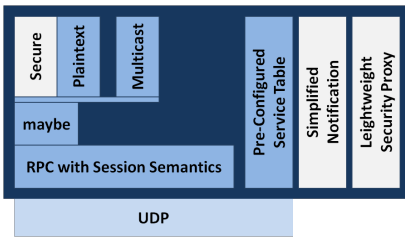

Figure 7: Functional Components of the Minimum Specification

The minimum specification is restricted through the processing power and especially memory of its hosting ECU. The key question is: What is the minimal communication subset needed even by a small ECU? The basic requirements for interactions with the larger middleware implementations are a common serialization and a basic Remote Procedure Call (RPC) functionality. We assume TCP transport not always possible on a small ECU due to memory size. Therefore, transport is executed over UDP which raises two challenges. The middleware must either implement a session semantic, or delegate the responsibility for execution to the calling application. If the application needs connection-orientation, the applications must manage the session itself or the middleware must provide an according mechanism above UDP.

Concerning service management on small-footprint devices dynamically maintenance of remote services is not desired. The service management process running in parallel to the actual application can critically slow down the device and fast start-up times are crucial. Therefore a hard-coded, preconfigured service table is used on such devices.

The notification mechanism is uncritical if the small-footprint device takes the role of a data consumer. If it is in the role of the service producer though, the challenges are the same as for service management described above. Instead of dynamically maintaining a list of consumers, the small-footprint device can use a pre-configured table. This can lead to a problem: Imagine, one of the consumers is sleeping. A producer that tracks the states of its consumers would first wake the device and afterwards send the data, while the consid-

ered small-footprint device cannot notice that the consumer is not reachable. If this problem can occur for a certain use case, one solution is to incorporate a more powerful delegate that never sleeps, e.g. the Head Unit, which serves as a forwarder to the actual consumer.

The restricted processing power reduces options for addressing security goals. Prototypical implementations give evidence that advanced security mechanisms based on asynchronous keys are not feasible on small ECUs. However, symmetric key encryption can be used for in-car communication. The Advanced Encryption Standard (AES) has proved feasible for symmetric encription on small devices [21]. In addition to that, the network topology must ensure that a small device is protected by a capable powerful one which serves as a security guard against the more likely external attacks.

4. SAMPLE SOLUTION

This section presents practical experience on how the identified requirements and the derived concept are addressed using an open source solution – Apache Etch.

4.1 Apache Etch

Referring to its web presence, "Etch is a cross-platform, language- and transport-independent framework for building and consuming network services. The Etch toolset includes a network service description language, a compiler, and binding libraries for a variety of programming languages. Etch is also transport-independent, allowing for a variety of different transports to be used based on need and circumstance. The goal of Etch is to make it simple to define small, focused services that can be easily accessed, combined, and deployed in a similar manner. With Etch, service development and consumption becomes no more difficult than library development and consumption. [1]"

It is industrial practice to commonly develop and use non-differentiating components among competing OEMs. The larger the user group, the more stable and cheaper the components get. We see an automotive communication middleware non-differentiating, consequently we assume Open Source Software (OSS) the best practice to get there. As the automotive industry still lacks practical experience with open source software, the closest approach is to adapt an existing project. Etch is maintained and developed as an Apache Incubator project and available as Open Source Software (OSS) under Apache License 2.0

Why we chose Apache Etch:

- Being an open source software project under a non-restrictive license makes it favorable for industrial adoption.
- Apache Etch provides an object-oriented interface definition language. Two communication partners are both specified in one definition file which follows the analogy of a contract. This concept matches the automotive practice well.
- It already comes with an efficient RPC mechanism running on various platforms and
- uses an efficient binary serialization format that is still resistant to changes along the path of interface and application evolution.

4.2 Required Extensions

The current state of the Apache Etch project still lacks some extensions that will be necessary to qualify the solution for usage in the automotive domain.

Etch was originally developed on top of reliable, connection-oriented TCP transport. Although it claims transport independence, design decisions must be taken when switching to connection-less UDP transport concerning sessions. When an Etch client connects to a server, it first establishes a TCP connection. That connection is kept open even after the first remote procedure call is complete. That not only simplifies the next call from client to server, it also allows the server to call procedures on the client and thus providing bi-directional communication. Let us assume we have an application using Etch over TCP and we want to exchange the underlying transport with UDP. If the Etch UDP implementation realizes the session handling, we can re-use the existing application code without modification. On the other hand, we introduced the UDP version mainly because a small-footprint device must not be slowed down with session maintenance. If we choose to implement a UDP binding without session handling, it is no longer compatible with TCP-based applications and we are restricted to unidirectional communication. Both options have been developed and are being contributed to the open source community.

Etch does not include native service management, so we have to add a mechanism for that task. There are two different approaches to do so: The first one is to adapt an existing qualified service management solution like Zeroconf [7] [19], for instance, in Apple Inc.'s implementation Bonjour. An already accomplished analysis of Bonjour gives evidence that this solution would be applicable to the IP-based in-car network. The second approach is a service management extension within the Etch RPC mechanism, meaning that a standard Etch-service using ordinary Etch RPCs is used for service management. This way, it is possible to tailor the service management mechanism towards the requirements. We have already developed the latter approach and will also contribute this standard service to the Apache Etch project. A comparative evaluation of the two approaches is to come.

Etch was not originally developed for communication between embedded devices. We assume that it can be adopted to automotive-qualified ECUs. This proof must make two main points: Verification of the embedded-suitability of the solution and verification of interoperability between embedded-optimized and existing derivates. The embedded-suitability will be mainly evaluated towards memory consumption of the program code, the process, and the pure middleware performance in calls per second. We therefore implement an "Embedded C-Binding" with reduced functionality according to the above presented minimum specification and high-degree platform independence. A first version of this derivate has currently been ported to an automotive FADO MPC5668 ECU running eCos OS. For the simplest possible Etch service with only one single method *void* $f()$,

the compiled program code size is under 65 kB – including linked communication and i/o libraries, which account for the largest part of this value.

5. CONCLUSION AND FUTURE WORK

In this paper, we first motivated a migration from the current, heterogeneous in-car network with special automotive technologies to an integrated IP-network using standard technologies like Ethernet. While many low-cost ECUs cannot be easily migrated to IP, we identified several candidate ECUs for a sensible migration. Furthermore, we sketched the need for a general IP-based middleware to be used by all in-car applications. The special requirements in vehicles led to the insight that we need a scalable solution that allows for several derivates of the middleware framework that differ in footprint and capabilities, but nevertheless are fully interoperable. Our proposed solution is based on an open source project, Apache Etch. Several extensions have been identified to qualify Apache Etch for automotive use.

Our future work includes the further extension of Apache Etch to prove the applicability of our solution even on very small embedded devices and automotive operating systems. Sample implementations of our middleware framework include prototype devices of the abovementioned migration candidates, namely IP-cameras, an IP-amplifier and an IP-radio module. We will also integrate these devices to our prototype "IP car" to give a comprehensive picture of the future, integrated in-car communication based on IP.

Furthermore, we plan to integrate our proposed solution with Autosar (AUTomotive Open System ARchitecture) [17] [5] which is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. Autosar describes communication interfaces for several technologies, including Ethernet. The Autosar-internal representation of a "Protocol Data Unit" (PDU) is independent from an actual bus system. Yet, a naive mapping of this PDU to IP does not yet allow for the full functionality according to our requirements. Our first goal is to find a wire format that is both Autosar-compatible and still can be mapped to the functionality of an existing middleware like Apache Etch.

6. ACKNOWLEDGMENTS

The tasks described have been executed within the SEIS – Security in Embedded IP–based Systems – project. The research project explores the usage of the Internet Protocol (IP) as a common and secure communication basis for electronic control units in vehicles. The project is partially funded by the German Federal Ministry of Education and Research (BMBF) (support codes 01BV0900 – 01BV0917). Partners involved in the SEIS project are Alcatel-Lucent Deutschland AG, Audi AG, Audi Electronics Venture GmbH, BMW AG, BMW Research and Technology GmbH, Continental Automotive GmbH, Daimler AG, EADS Deutschland GmbH, Elektrobit Automotive GmbH, Infineon Technologies AG, Robert Bosch GmbH, Volkswagen AG, the University of Erlangen-Nuremberg, the Karlsruhe Institute of Technology, the Technical Universities of Chemnitz and Munich, and the Fraunhofer Institutes for Communication Systems ESK and for Secure Information Technology SIT. The project is coordinated by BMW Research and Technology GmbH in Munich.

7. REFERENCES

[1] Apache Etch homepage. `https://cwiki.apache.org/ETCH/home.html`.

[2] A. L. Ananda, B. H. Tay, and E. K. Koh. A survey of asynchronous remote procedure calls. *SIGOPS Oper. Syst. Rev.*, 26:92–109, April 1992.

[3] A. D. Birrell and B. J. A. Y. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984.

[4] R. Bruckmeier. Ethernet for Automotive Applications. `http://www.freescale.com/files/ftf_2010/Americas/WBNR_FTF10_AUT_F0558.pdf`, 2010. Freescale Technology Forum, Orlando.

[5] S. Bunzel. AUTOSAR - the Standardized Software Architecture. *Informatik-Spektrum*, 34:79–83, 2011. 10.1007/s00287-010-0506-7.

[6] CAN-CIA. CAN Specification. `http://www.can-cia.org/`.

[7] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses, RFC 3927. `http://www.ietf.org/rfc/rfc3927.txt`, 2005.

[8] M. Glass, D. Herrscher, H. Meier, M. Piastowski, and P. Schoo. SEIS – Security in Embedded IP-based Systems. *ATZelektronik worldwide*, 2010-01:36–40, 2010.

[9] IEEE. IEEE P1722.1/D0.12 - Draft Standard for Standard for Standard Device Discovery, Connection Management and Control Protocol for IEEE 1722 Based Devices. `http://grouper.ieee.org/groups/1722/1/`, 2010.

[10] IEEE. IEEE P1722/D2.4 - Draft Standard for Layer 2 Transport Protocol for Time Sensitive Applications in a Bridged Local Area Network. `http://grouper.ieee.org/groups/1722/`, 2010.

[11] H. Lim, L. Völker, and D. Herrscher. Challenges in a Future IP/Ethernet-based In-Car Network for Real-Time Applications. In *Proceedings of the 2011 ACM/EDAC/IEEE Design Automation Conference (DACÍ1)*, San Diego, USA, June 2011.

[12] H.-T. Lim, K. Weckemann, and D. Herrscher. Performance Study of an In-Car Switched Ethernet Network Without Prioritization. In T. Strang and al., editors, *Nets4Cars/Nets4Trains 2011*. Springer-Verlag, 2011.

[13] MOST-Cooperation. MOST Website. `http://www.mostnet.de/`.

[14] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert. Trends in Automotive Communication Systems. *Proceedings of the IEEE*, 93(6):1204–1223, 2005.

[15] M. Ruff. Evolution of Local Interconnect Network (LIN) Solutions. In *IEEE 58th Vehicular Technology Conference. VTC 2003-Fall*, pages 3382–3389 Vol.5, 2003.

[16] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications, 2003.

[17] Simon Fürst and al. AUTOSAR - A Worldwide Standard is on the Road. 14th International VDI

Congress Electronic Systems for Vehicles, Baden-Baden, 2009.

[18] B. H. Tay and A. L. Ananda. A survey of remote procedure calls. *SIGOPS Oper. Syst. Rev.*, 24:68–79, July 1990.

[19] C. N. Ververidis and G. C. Polyzos. Service discovery for mobile ad hoc networks: A survey of issues and techniques. *IEEE Communications Surveys and Tutorials*, 10(1-4):30–45, 2008.

[20] K. Weckemann and B. Weyl. Aus der IT-Welt ins Auto – Sichere Kommunikation im Fahrzeug mit dem Internet Protocol. *Elektronik automotive*, (12):38–42, 2010.

[21] B. Weyl and al. EVITA Deliverable D3.2 Secure on-board architecture specification. `http://www.evita-project.org/`, 2010.