ICT-Architecture for Multi-Modal Electric Vehicles

Oliver Horst, Patrick Heinrich, Falk Langer
Fraunhofer Institute for Embedded Systems and
Communication Technologies ESK
Munich, Germany
Email: {oliver.horst, patrick.heinrich, falk.langer}@esk.fraunhofer.de

***Abstract*—This paper presents a new information and communications technology (ICT) architecture for future automobiles, which enables an easy and flexible modification of its functionality by end user, operating companies or manufactures. This enables software-based extensions of company-specific functionality (e.g. fleet management software) without the today's need of additional hardware. Through that a car is adaptable and multi-modal, because of the change of functionality within seconds. We call this kind of vehicle a *software-defined car*. New possible business models are discussed, which occur through software-defined cars. Companies for example would be able to share cars with other companies to adapt the number of vehicles to the current demand or third parties could develop and sell functionality for the car. The ICT architecture as core of that kind of multi-modal car defines three different levels of adaption with different restrictions depending on the access rights of the different interest groups. An adaption is possible from the human machine interface (HMI) to the driving functionality like the acceleration degree. The ICT architecture enables the influence-free execution of applications with different safety levels, which enables aggregation of control units within the electric and electronic (E/E) architecture as additional advantage. Finally the prototyping vehicle is presented, at which the ICT architecture will be realized and demonstrated.**

Keywords*—*3.0.V. System architectures, integration and modeling; 3.II.0.I. Architecture; 3.II.IV. Real-time and embedded systems; 3.V.VI. Multiprocessor Systems; 4.II.XI. Software Architectures; 11.m.I. Business

I. Introduction

At the moment one of the basic problems of electric vehicles is their acquisition costs. Even if the costs during operation are significant smaller, the total cost of ownership (TCO) seems to be higher than for vehicles with combustion engines. As long as the production costs cannot be reduced, increasing the utilization of each electric vehicle is one possibility to enlarge their competitiveness. This paper presents a new business and vehicle concept, which intends to increase the utilization of the vehicle and therefore reduces the TCO. In focus is the commercial usage and the ownership by enterprises. The key aspect for these concepts is the multi-modal usage of one electric vehicle, which means the easy switching between the "modes" of the vehicle enabled by different applications, and the possibility to share the vehicle between different companies.

The goal of the new concepts is to increase the flexibility and adaptability of the vehicle and its services, which are offered to the driver. For example the same vehicle shall be usable for taxi-services, car-sharing, and city logistics. For each of these intended usages, however, specific services and applications has to be provided. To reach this goal a personalization is necessary concerning the driver, its company and the specific area of operation. Nowadays, the services and applications provided by car are more or less static and designed by the car manufacturer. This fits well to the private usage of cars were the needs of consumers are relatively homogeneous and constant, which is not the case for business use cases. The approach presented in this paper strikes out in a new direction by dividing this sovereignty to the three stakeholders: the car manufacturer, the operator and the user of the car. Each of these three parties should be able to modify the cars behavior and the services offered to the driver.

In modern vehicles most of the new functionality is realized within software, which means modifying the software means modifying the range of applications. Thinking this approach through, the objective of a totally software-defined car came up. Basically that means that the manufacture only provides vehicles with the basic driving functionality. All other functionalities of the car – not essential for driving – are provided by resellers, operators or the user. Reaching this objective would create a flexible vehicle for every use case, which is able to fulfill the specialized needs of different users by simply changing the software. This concept of providing a piece of hardware with a basic set of applications, where users can easily extend the functionality is already exercised in modern smartphones. In difference to smartphones, most of the functions within a car are in a way safety critical. Through that it is necessary to isolate applications with different safety levels from each other to enable the execution without influences. Such systems are called *mixed-critical systems*. Solving the challenge of mixed-critical systems enables the aggregation of several electronic control units (ECUs) to a single control unit. Using less ECUs additionally reduces the price of electric vehicles, i.e. the TCO.

The paper presents a new layered software architecture that enables the described usage of the car and ensures the correct and reliable functionality of electric vehicles. The benefits and usage of this new architecture is shown within a case study based on a prototype for a new small electric vehicle. Since the proposed application of this new software architecture is the usage in a simple and small city vehicle, the focus is not on maximum functionality with maximum of comfort as in premium cars. The new architecture shall provide a maximum flexibility by minimal cost.

The Paper is structured as follows, Section II presents and discusses the new business model, which is the motivation for the proposed software architecture. In Section III a short overview about the related work and the state of the art of software and software development for in-vehicle applications is provided. Section IV introduces the basic concept of the

proposed new ICT-Architecture. Core part of the technical application of that ICT architecture is a new deterministic communication scheme that is introduced as deterministic message passing within Section V. To demonstrate the correlation between the business concept and the new ICT-Architecture in Section VI a case study for a small electric vehicle is presented. The paper closes with a conclusion and future work in Section VII.

II. Business Model

New business models are necessary to reduce the cost for (electric) mobility by maximizing the utilization of every single vehicle and splitting the purchase price and the operational costs. Sharing the costs means sharing the vehicle, which seems to be simply possible, because most of the cars are unused outside business hours and even during a normal business day the company cars are not utilized all the time. The problems are coming to light looking at the details, because most of the company cars are specialized (e.g. taxis need taximeters) or equipped with company-specific functions (e.g. a reservation or location discovery service for the company fleet management). This specific functions are normally installed within the car by installing an additional hardware, which is placed at the car trunk or the footwell. Beside the restricted access due to the car manufacturer, an installation of additional software is normally not possible, because the distributed ICT architecture does not allow this by design. This simplifies the integration of software and the isolation of software between each other, but means that car sharing between companies is not possible, because every vehicle need additional installed hardware to realize the specialized functionality. Beside the need of this functionality, companies normally want to show the affiliation of the car, which means very often the easily visible company name on the car. This is realizable using electronic ink displays, which are good readable even in sunlight and have a low energy consumption. If there is no need for presenting the company name, the display is usable for advertisements and enables additional earnings. Summarized, this means from the manner of using vehicles like these, there is no difference concerning functionality and usability.

Through that new business models are possible. Especially software for automobiles is a large field of development, where companies would be able to sell own software functionality. And operating companies are able to extend these cars with individual functionality using own software applications. This is a typical after-market sector, which is currently not very common within the automobile industry. Companies using this kind of cars, e.g. for their fleets, are able to save money, because car sharing between companies is possible. This means cars a shared with other companies, when they are normally unused. This means an increase of utilization and a reduction of costs – especially if vehicles are just needed during peak hours. The reason for this is the disappeared need of equipping company's fleet to handle the maximum demand of vehicles. Companies are able to book vehicles per hour or minute such as private persons, which use (private) car sharing. Of course, this is no opportunity for companies who's cars are in operation 7 days the week, 24 hours a day. However, most companies have unused times of their cars or peak hours. Here the potential exist to reduce costs by getting vehicles on demand. And also during driving it is possible to earn money, because of changing advertisements at the vehicle allowed by electronic inc displays. Especially location based advertisement would be profitable, which means showing advertisements of nearby companies, e.g. taxis could display advertisements during their wait time at the taxi-stand. This kind of advertisement is realizable using position data from the built in GPS sensors.

As example, a possible day of such a cross-company shared vehicle is presented in the following: Early in the morning local newspapers are delivered. An application shows the driver the next destinations for the newspaper and the best driving route to save time and energy. Some hours later the same vehicle is used within a company's fleet, where the employees use the car for business purpose and need a possibility to book the car. And if the car is used for a private trip, employee and company need an application to differentiate the trips in a way that also the tax office accepts it. In the evening the car is used as taxi, which means the need for a taximeter and also the possibility to get driving orders from the taxi control center. During waiting periods the taxi driver uses the displays at the outside of the car to present location based advertisement.

Today, this kind of business models are realizable within software, but the current ICT architecture does not provide the necessary flexibility. That is the reason we present a new ICT architecture. Through that, every company is able to extend the functionality of the vehicle to customize it. Even the user itself is able to personalize the vehicle, if the company-specific application allows these extensions. The ICT architecture presented in Section IV enables that kind of usage and hereby enables the described business case.

III. Related Work and State of the Art

In the last years, there is a rapid growing of software defined functions within the car [1]. This trend is primary visible within premium cars, which are the flagships of each car maker and demonstrate the technologically feasible. The two most important drivers of software functions are the infotainment domain and the area of driver assistant systems. Especially the ICT infrastructure of the infotainment domain has large drawbacks in comparison to modern consumer electronics. This is a result of longer development cycles and the need to ensure a correct protection, because consumer electronics are not used within safety critical applications.

This leads to the effect that small and low cost cars normally do not contain much infotainment equipment. People often use their smartphones within those cars as compensation for the missing infotainment system. The usefulness of this approach is limited by the difficult and restricted connection possibilities between car and smartphone. Most of the current solutions try to integrate the smartphone within the car. To do so the car must be equipped with expensive technologies like touch screens and infotainment controllers. Another approach is the complete replacement of the in-car infotainment by the smartphone, but there are currently no simple solutions that car services and internal data can be used within a smartphone. If this would be possible, one could build a cheap car that can be easily equipped with the newest technology from the consumer market.

The first upcoming steps towards such a solution of integration are visible within some software projects or standards, which are driven by car makers. One important development

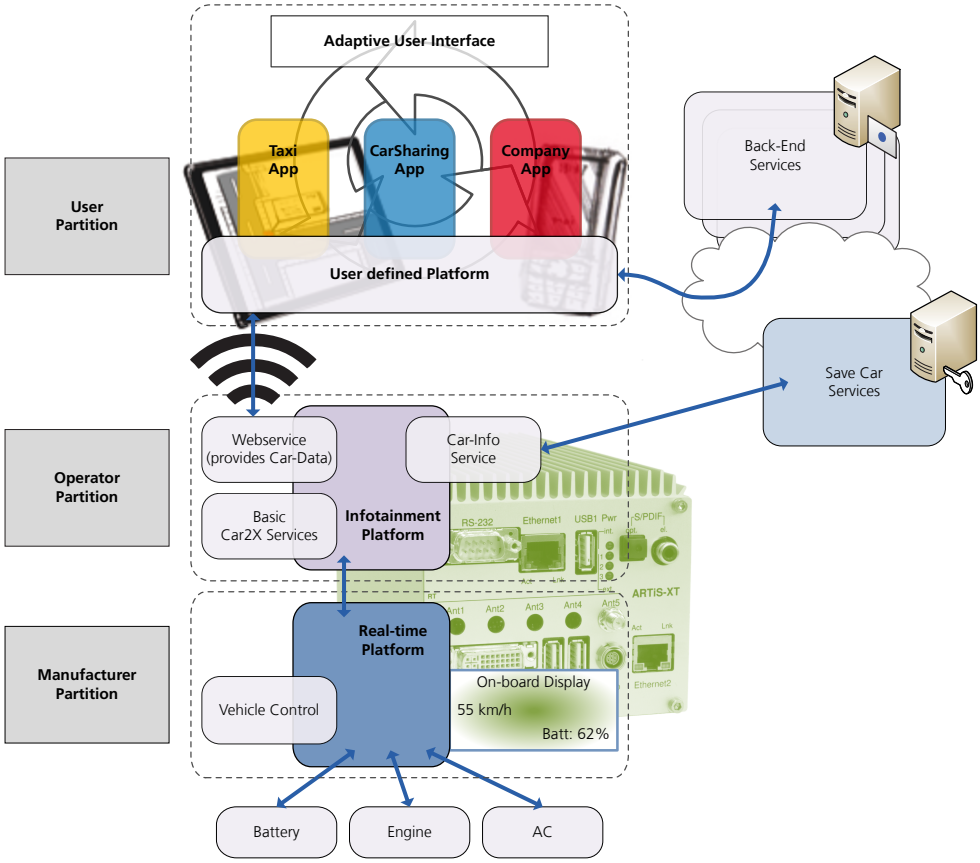

Figure 1. Overview on the ICT architecture with the three partitions for the stakeholders: manufacturer, operator, and user. Each partition runs on its own platform and is isolated through strictly defined interfaces from the others.

project that deals with this kind of problem is GENIVI [2]. This is a Linux-based open-source in-vehicle-infotainment solution. Within the GENIVI project there is a working group that works on the establishment of a web service interface for the GENIVI platforms that offer data to other applications. Another important step is the opening of AUTOSAR to the TCP/IP standard. Within the newest release AUTOSAR will support the SOME/IP protocol [3], [4] that enables a remote procedure call over TCP/IP connections. This work-in-progress shows that there is a paradigm shift from proprietary protocols to standardized protocols inside the car.

The ICT architecture presented in this paper opens the car to external devices, which results in an paradigm shift. Realizing this ICT architecture means in an extreme case no longer the integration of the smartphone within the car, it means the car is integrated to the smartphone. This is due to the fact that essential functionality is executed on the smartphone and the car is a kind of peripheral. Of course, this view is a bit black and white, because especially the safety critical functionality need to be ensured. This aspect is also considered within the presented ICT architecture.

IV. ICT-Architecture

Today, the electric/electronic (E/E) architecture of cars follows the principle of one electronic control unit (ECU) per function. This approach greatly simplifies the integration of different components and the overall certification of the vehicle. However, we assume that for small electric vehicles this traditional concept has to evolve to a somewhat leaner E/E architecture, to provide all the flexibility and cost savings we discussed in the previous sections. Electric vehicles are subject to high pressure of cost anyways and small versions of them simply do not have enough space for up to 90 electronic control units, as they are placed in todays traditional cars. Thus, we postulate that the E/E architecture of small electric vehicles will evolve to an architecture with only one, or at most a few central electronic control units. This central electric control unit (CECU) shall be flexible and adaptive enough to cope with the interests of all three stakeholders of the car, while it incorporates all functionality of all ECUs present in a traditional E/E architecture. Even updating the software in the field, as well as extending the functionality of a car with new software shall be possible with the new E/E-architecture and the new CECU respectively. This openness, however, poses strong challenges upon the software architecture of that control unit. Existing software/information and communications technology (ICT) concepts cannot be deployed to such a consolidated, but still flexible and adaptive CECU right away. The software is designed with exclusive access to all hardware resources in mind and expects a closed environment where all possible interferences were known at design time of the software.

We propose the software architecture shown in Figure 1 as possible solution for consolidated electronic control units. It is a flexible ICT architecture that on the one hand provides flexible

and highly adaptable environments for the three different stakeholders: the manufacturer, the operator and the user of a car. And on the other hand strongly encapsulates each of the environments and guarantees a safely execution of all software components according to applicable regulations. In accordance with the common sense in the field of virtualization technologies we call each of the stakeholder environments a *partition* of the overall system. Each of the three system partitions has its own properties and usage scenarios, which reflect the requirements of the intended audience. The *manufacturer partition* provides the core functionality of the car. It executes tasks such as the engine control, the body domain control, and driver assistance systems. Generally, the manufacturer partition hosts safe, trustworthy, and probably even vital software components that require a certain real-time behavior. The *operator partition* hosts software components that provide services to the car or its user. Such services could be e.g. a provider of car status information, a audio/video sink provider, but also components like a taximeter or a electronic drivers logbook. All those software components and services have in common that they need to be protected against manipulation and do not assume any particular real-time behavior. As discussed in Section II the operator partition can be used to extend the ICT system of the car with additional functionality precisely tailored to a specific use case, without influencing the core functions of the car. The applications within the *user partition* are build upon the services provided by the operators partition and the back-end services to provide a front-end for the different car functionalities and use cases (e.g. taxi-app or car-sharing-app – cf. Figure 1). Thus, the only difference between conventional smartphone apps and the user applications from our concept is the employed interface.

The advantage of our concept is that the user partition, and with it the human machine interface (HMI) and the infotainment system, is completely decoupled from the in-car system components (cf. Figure 1). Each user is thus offered with the opportunity to use or even build his very own user interface perfectly suited to his habits and needs. Furthermore, by relying on user hardware for the infotainment system and the HMI, we implicitly solve the problem of adapting the in-car infotainment system to the fast paced development cycles of modern consumer electronics. The user can benefit from the latest features of his hardware and use the car only as a "simple" input/output device, while the core functions of the car can be developed with the same conscientious approach as before. Thus, the final functionality of the car is just defined by the users software; we call it the *software defined car*.

Figure 2 illustrates our technical application of the presented ICT architecture. In Figure 2a our first prototype is shown and in Figure 2b our planned solution. The prototypical implementation is based on our automotive prototyping platform ARTiS-XT [5], a flexible electronic control unit suitable for in-car applications based on two processor-boards: a Freescale MPC5554 for real-time applications and a Intel Atom Z520PT for infotainment applications. The advantage of a prototypical implementation on two processor boards is that initially no efforts are required to isolate the manufacturer and the operators partitions from each other, as they are already separated physically. Nevertheless, the two boards need to communicate with each other. We solved this with an interprocess communication (IPC) that utilizes a isochronous USB connection between the two boards.

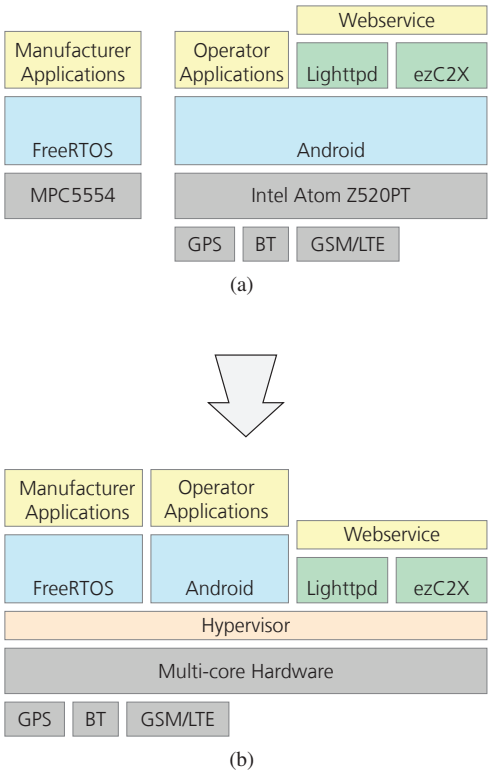

Figure 2. Overview on the planned ECU transition from a two processor board based version (a) to a multi-core platform based version (b). Shown are the individual components of our technical application of the ICT architecture presented in Section IV and their placement.

The manufacturer partition is classically realized on the PowerPC based real-time platform and as operating system we utilize FreeRTOS. The operator partition is deployed on the Intel Atom board, here we use Android as operating system. The interconnect to the user partition, on e.g. a smartphone or tablet, is set up through a web service that runs on top of a Lighttpd[1] instance within the operator partition. The connection itself is established via HTTPS over Bluetooth with help of the Personal Area Networking Profile (PAN). Each operator application can provide its own web service to extend the usable feature set of the user applications, however, one general vehicle web service is provided by default. The default web service provides all basic information about the car and its current state, examples are the current speed, the GPS position, or charging status. Furthermore, the default web service offers control over the user controllable hardware of the car, like the windshield wipers, the speakers, or the air conditioning, however not over the safety critical parts. These parts are strongly separated within the manufacturer partition. To simplify the development of web services and to give other applications in the operator partition the capabilities to access the vehicle functions and information we employ our car-to-X framework ezCar2X [6].

In a second step we plan to utilize virtualization concepts and techniques to integrate both the manufacturer and the operator partition on one multi-core platform (cf. Figure 2b). For the integration purposes we already built and presented a flexible operating system concept [7], which we like to extend with virtualization support. The flexible operating system

[1]Open-source web server – http://www.lighttpd.net/

concept bridges the gap between symmetric multiprocessing (SMP) and asymmetric multiprocessing (AMP). The design provides for a single-core operating system which is deployed in an AMP configuration. Each core executes its own instance of the kernel; there are no shared data structures within the kernel. This significantly reduces run-time complexity, and thus facilitates analysis of timing behavior and resource usage. The kernel of our operating system is flexibly configurable, the overall concepts is expandable by multiple front-ends like e.g. a POSIX compatibility layer, or an AUTOSAR RTE. Thus, our concept is not purposed to replace existing concepts, but to complement them and make them more flexible. To enable cooperation of the independent OS kernel instances we provide a message-based mechanism for inter-process communication (IPC). For the present application of a electronic control unit, with a mixed criticality software stack, a reliable message transfer algorithm is needed to guarantee a reliable spatial and temporal isolation of the manufacturer and operator partition on a multi-core platform. We developed such an algorithm in form of a deterministic message passing approach, which is discussed in the following section. With the reliable communication mechanism in place and the virtualization support at hand it is even possible to provide individual containers within the manufacturer's and operator's partition for each individual software supplier [8]. Thus, suppliers could continue their independent development processes and the manufacturer would still retain the sovereignty over the software integration in the manufacturer partition.

V. Deterministic Message Passing

To support a reliable communication mechanism, we refined the original operating system concept presented in [7] by introducing categories for tasks and processor cores. Tasks are categorized into five types: real-time (hard and soft), best-effort, kernel(-service), or interrupt service routines (ISR). Processor cores are differentiated into: one communication core and several processing cores. The communication core is a dedicated core that processes all incoming interrupt requests and manages the communication infrastructure. The processing cores, on the other hand, simply execute tasks from different partitions and of different criticality. However, both processor core types execute best effort tasks during spare time. Figure 3 illustrates this task assignment model.

On top of the extended architecture we propose a time-triggered and message based communication scheme: the deterministic message passing. A time-triggered communication scheme has the advantage of being easier to certify and prove correct [9]. In our setup, the communication core determines the sequence of task execution and message exchange phases and thus the timing of the communication, which is also illustrated in Figure 5. The task execution on the processing cores is split up into schedule frames, which are oriented to the message exchange phases on the communication core. A schedule frame itself consists out of one schedule container per system partition and a single message exchange phase at the end. A schedule container in turn is an excerpt from the schedule plan for the tasks of the system partition that is associated to the schedule container. This kind of scheduling enables a fine grained control over the resource assignment per system partition and is known as hierarchical scheduling [10], [11].

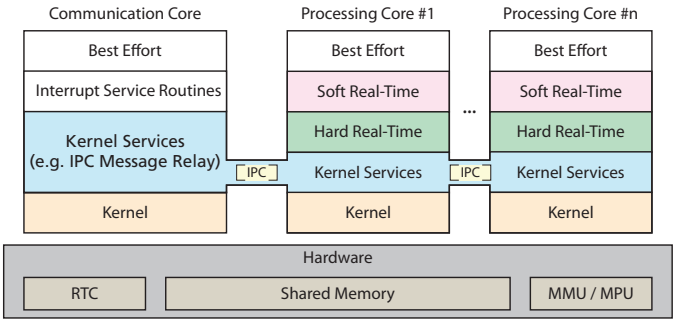

Figure 3. Illustration of the task and processor categories, as well as of the task-assignment model of the proposed deterministic message passing architecture.

The message exchange phase at the end of each schedule frame ensures the transmission of all message that were send during that communication interval. A communication interval is defined as the time between two message exchange phases, which matches a schedule frame except for the final message exchange phase (cf. Figure 5). Transmitted messages are always delivered in the second schedule frame that follows the frame in which they were send. Thus, when a task sends a message in schedule frame $(n-2)$ this message is processed by the communication core in schedule frame $(n-1)$ and delivered to its destination task or core in schedule frame n. The overall process that leads to this delay can be tracked in Figure 4.

As stated before, the communication core determines the timing of the communication, this is done by actively triggering the processing cores for new messages within the message exchange phases. Precisely, we differentiate between the following five phases, as illustrated in Figure 4:

1) *Send phase.* Each message an application sends is composed from destination information (core or task), the payload, and credentials. Ready assembled messages are placed into the core's send buffer. The credentials are later used in the *inspection phase* to legitimate the message transfer.
2) *Exchange phase.* The communication core acquires the messages from the send buffer of one processing core after the other and replaces the current send buffer with an empty one and the current receive buffer is supplemented with the composed messages for that core from the previous *composition phase*.
3) *Inspection phase.* After acquiring all message buffers the communication core checks the individual messages for validity and legitimation. In case the provided credentials do not legitimate a message transfer the message is discarded.
4) *Composition phase.* After verifying the messages the communication core composes an individual receive buffer per processing core, containing all messages addressed to that specific core or tasks that run on it.
5) *Receive phase.* Finally, after a second *exchange phase*, the messages are delivered and tasks on the destination processing core can access the transmitted information. In the proposed communication scheme, applications are not explicitly informed about the arrival of new messages. Instead, applications are intended to poll the receive buffer on their own if they expect a message

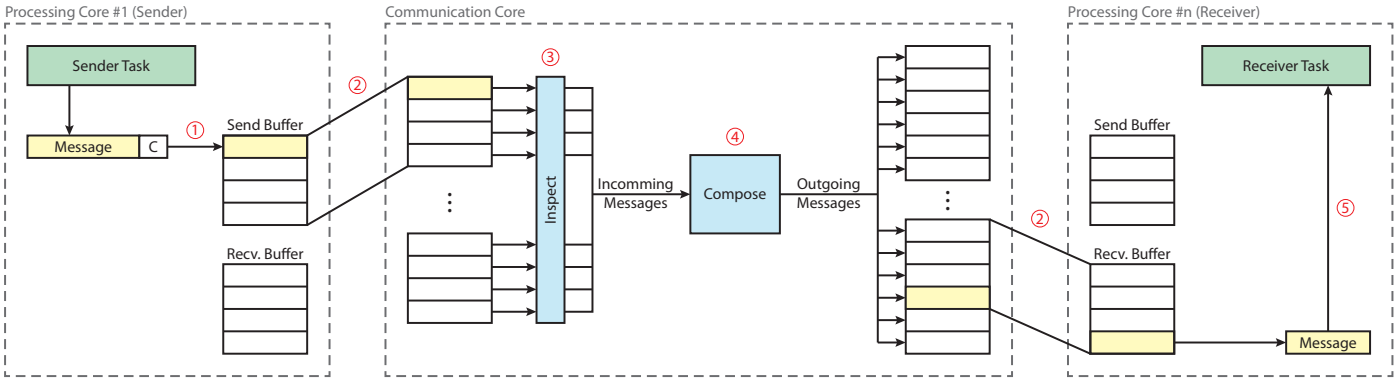

Figure 4. Illustration of the five-phases of the proposed deterministic message passing concept, exemplified by the transfer of a single message from processing core #1 to processing core #n. The five phases are: 1. send phase, 2. exchange phase, 3. inspection phase, 4. composition phase, 5. receive phase.

to be delivered.

As a consequence of the described process, processing cores can continue task execution concurrently, while the messages are exchanged by the communication core. Ideally, acquiring the send buffers and transferring the receive buffers could be realized with a single atomic operation that interchanges the buffers. Thus, causing no locks or waits on the processing cores. However, this depends on the properties of the targeted hardware platform.

Various researchers already presented approaches to execute software with concurrent operations in a predictable and repeatable manner. For example, [12] and [13] suggest runtime environments that guarantee a deterministic execution of concurrent programs with the help of a code instrumentation done by the compiler [12], or with the help of a specification language and a suitable virtual machine [13]. Both approaches have in common that in contrast to our approach they rely on the shared memory programming model. However, a reliable isolation of system partitions in a shared environment in space and time cannot be guaranteed with the shared memory approach. In contrast, our time triggered message passing programming model allows commitments to upper bounds for run-times and issue times of individual tasks. Other researchers focus on solving the same issue in hardware by extending the instruction set of the processor with specific deadline management instructions [14]. As an alternative approach, researchers suggest to build a time-triggered communication on-chip to allow a deterministic communication between the individual components of a System-on-a-Chip platform [15]. Our approach, however, focuses a solely software based solution which is suitable for commercial off-the-shelf (COTS) hardware. We think that this is an important point in terms of the reduction of the overall production costs of a vehicle.

The approach of dedicating a single processor core for specific tasks, like our communication core, is also a well known concept, especially to reduce the costs of context switches and interrupt handling in virtualized environments [16]–[18]. Nonetheless, the presented concept of a deterministic message passing is the first that is especially suited for mixed-criticality systems and simultaneously guarantees upper bounds for the time until a interrupt is processed, as well a deterministic execution of concurrent tasks.

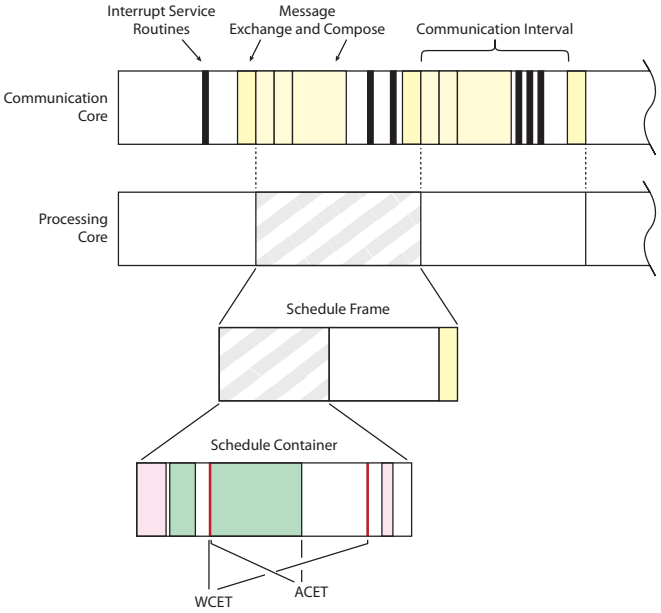

Figure 5. Link between the schedule plan of the communication core with that of a processing core. Shown are the different message exchange phases on the communication of which one (highlighted with a slightly darker yellow) is correlated to the shown processing core. On the processing core the relationship between schedule frames, schedule containers, and the final message exchange phase is shown. After the three message exchange phases on the communication core a message composition phase follows. All yellow marks correspond to communication related tasks, the black marks correspond to interrupt service routines, and the green and red marks correspond to soft and hard real-time tasks.

VI. Case Study: Adaptive City Mobility

The ICT-architecture described in the previous sections is currently being realized within the research project “Adaptive City Mobility” [2]. The objective of this project is the realization of a small electric vehicle, which distinguishes itself by providing a simple and flexible design. The vehicle itself is manufactured using lightweight materials, i.e. carbon fiber reinforced plastics, and has a modular construction to be flexible even with the vehicle body. Through that different variants are feasible, e.g. for passenger transport, city logistic or other

[2]http://www.adaptive-city-mobility.de

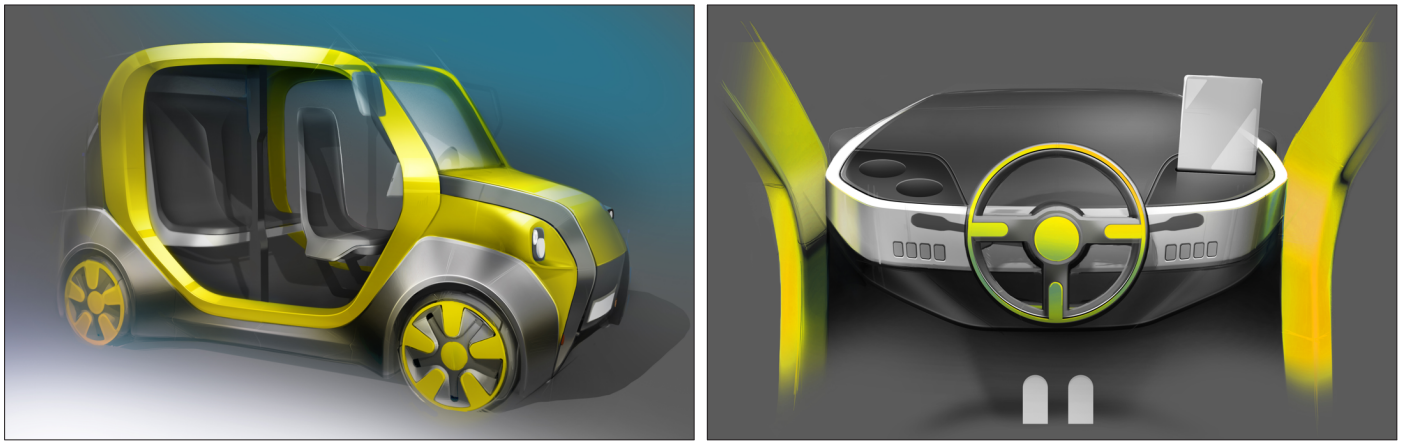

Figure 6. Design study of the multi-modal vehicle, which shows a version without doors. The design of the car is derived from the saint Bernard dog and shall indicate the helpful character of the vehicle. The interior is simple and well structured to enable an easy use for everybody. Just the absolutely necessary driving elements are realized within hardware - all other functionality is realized within software.

applications. The drive train of the vehicle consists of two electric engines at the rear axle, which are positioned near to the wheels. Electrical energy for the engines is provided by a rechargeable battery, which is also designed modular. This enables the possibility to equip the vehicle with more or less capacity depending on the trip or the load of the vehicle. To realize that modularity the battery consists of up to 8 identical battery modules, which are individually replaceable. However, the main advantage of this realization of the battery is the possibility to “refill” the battery within minutes, because it is not necessary to wait until the build-in battery is recharged. Every module weighs about 10 kilogram, so that a change of battery modules by hand is possible. This is done at specific battery exchange stations. Nevertheless, it is still possible to charge the battery using a recharger cable.

The core of the vehicle is the open information and communication technology (ICT) architecture that enables the flexible and multi-modal usage of the electric vehicle as described in the sections before. A central electronic control unit (CECU), which uses a multi-core processor, enables this ICT architecture with multiple virtual partitions that ensure the integrity of the safety-relevant software functions. User interface and user interaction are realized by an app on the personal smartphone or tablet computer, which is connected to the vehicle (e.g. the CECU) and placed near the steering wheel of the vehicle. The central electronic control unit is also connected to backend servers to exchange information, for example the current position or other application specific information. Applications installed at the CECU and communicating with the user tablet/smartphone and the backend servers are for example a taxi or car sharing application, as visualized within Figure 1.

To show the different functionality of the vehicle, different applications will be realized within the project. Using the vehicle as Taxi, the driver needs to know the next driving orders from the taxi control center or the distance to the next battery exchange station. These functions are realized and placed at the Android partition of the central electronic control unit and the user interface at the smartphone or tablet of the driver. This enables the taxi control center to get information direct from the vehicle, such as speed, state of charge of the battery or location of the vehicle. Changing the mode of the vehicle, for example after the shift of the taxi driver, to the mode ”Car Sharing”, other functions are necessary. A car sharing customer wants to book a car with her/his smartphone and after that wants to entry and start the car using the smartphone. This means the car has to send its position to the back-end servers and has to open the doors and start after the valid identification of the customer. The communication between smartphone and vehicle is realized using near-field communication (NFC) and Bluetooth. Using the car within a fleet makes it necessary to differentiate between private and business trips, and also need an exact time scheduling for the vehicles. These functions are also placed at the central electronic control unit and are activated after the mode change of the vehicle. Through that every operating company (e.g. taxi companies, car sharing services or even food delivery services) has the possibility to provide application- or company-specific functionality, which covers more than just the adaption of the graphical user interface, and allows the vehicle to be operated for different purposes. On the other hand, the user of the vehicle has the possibility to personalize the vehicle to her/his demand.

Figure 6 shows the current design of the vehicle. The vehicle is optimized to attain the objectives of the future applications with minimal weight, costs and energy consumption. Through that the chosen vehicle category is the German ’L7e’, which is the same category as quad bikes. The reduction of weight, cost and energy consumption is realized by implementing the previous described ICT- and E/E-architecture. This means that just one main electronic control unit is installed - beside some specialized controllers for example for engine control or battery monitoring. The reduction to a minimum is also reflected in the fact that there are just three seats within the vehicle. The reason for this is that most of the time not more than three persons are within a vehicle. And for using the vehicle as taxi it is required by law to be able to transport minimum two passengers, which results in the design of one central driver seat and two back seats. The battery modules are not visible within the shown design, but it is planned to place them at the bottom of the vehicle with access from the left and right side.

Also not shown are the displays for example at the outside of the doors, which enables to customize the vehicle depending on the current mode (e.g. Taxi) or to place advertisement. The displays shall be realized using electronic ink technologies, so that there is no additional energy consumption - except in the case of changing the display content.

VII. Conclusion and Future Work

An information and communications technology architecture for future automobiles was presented, which allows manufacturers, operating companies and end users to extend the functionality of the vehicle by adding software applications. This enables different new business models, e.g. sharing vehicles between companies, because every company is able to activate their functionality within second and satisfy the need of vehicles on demand. The different levels of the ICT architecture for the three interest groups were presented and a deterministic message passing to allow a reliable technical application on multi-core processors discussed. The next steps are the realization of the presented ICT architecture within the shown vehicle prototype. After that an in-depth evaluation of the architecture is possible and the results demonstrable.

Acknowledgment

The research leading to these results has received funding from the German Federal Ministry for Economic Affairs and Energy (BMWi).

References

[1] H. Ulrich Michel, "Taming multicores for safe transportation – ARAMiS in the automotive domain," presented at the Workshop for Integration of mixed-criticality subsystems on multi-core processors held in conjunction with the 8th HiPEAC Conf., 2013.

[2] GENIVI Alliance, "Automotive infotainment software architecture report," 2010. [Online]. Available: http://www.genivi.org/sites/default/files/GENIVI_IVI_Software_Architecture_Report.pdf

[3] AUTOSAR development cooperation, "AUTOSAR specification," v4.1.

[4] L. Völker, "SOME/IP - Die Middleware für Ethernetbasierte Kommunikation," *Hanser-Automotive Networks Spezial 2013*, pp. 17–19, 2013.

[5] Fraunhofer Institute for Embedded Systems and Communication Technologies ESK, "ARTiS-XT: Automotive telematics and infotainment prototyping system," Jan. 2013. [Online]. Available: http://www.esk.fraunhofer.de/content/dam/esk/en/documents/PDB_ARTiS-XT_en_web_neu.pdf

[6] ——, "ezCar2X: Streamlining application development for networked vehicles," Oct. 2012. [Online]. Available: http://www.esk.fraunhofer.de/content/dam/esk/en/documents/PDB_ezCar2X_en_web_neu.pdf

[7] O. Horst and A. Schmidt, "Operating system concepts for embedded multicores," in *Proc. of the embedded world Conf.* WEKA Fachmedien, 2014, p. 5.

[8] O. Horst and C. Prehofer, "Multi-staged virtualization for embedded systems," in *Proc. of the Work in Progress Session held in connection with the 37th Conf. on Software Eng. and Advanced Applicat. and the 14th Conf. on Digital Syst. Design*, ser. SEA-Publications of the Institute for Systems Engineering and Automation, E. Grosspietsch and K. Klöckner, Eds., no. SEA-SR-30. Linz, Austria: Johannes Kepler University, Sep. 2011.

[9] H. Kopetz, "Event-triggered versus time-triggered real-time systems," in *Operating Systems of the 90s and Beyond*, ser. Lecture Notes in Computer Science, A. Karshmer and J. Nehmer, Eds. Springer Berlin / Heidelberg, 1991, vol. 563, pp. 86–101.

[10] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proc. of the 4th Int. Conf. on Embedded Software*, ser. EMSOFT. New York, NY, USA: ACM, 2004, pp. 95–103.

[11] R. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proc. of the 26th Int. Symp. on Real-Time Systems*, ser. RTSS, Dec. 2005, pp. 10 pp. –398.

[12] T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman, "CoreDet: A compiler and runtime system for deterministic multithreaded execution," in *Proc. of the 15th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: ACM, 2010, pp. 53–64.

[13] C. Farcas and W. Pree, "Virtual execution environment for real-time TDL components," in *Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2007, pp. 93–100.

[14] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems," Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2013-172, Oct. 2013.

[15] M. Schoeberl, "A time-triggered network-on-chip," in *Proc. of the Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2007, pp. 377–382.

[16] K. Tian, Y. Dong, X. Mi, and H. Guan, "sEBP: Event based polling for efficient I/O virtualization," in *Proc. of the Int. Conf. on Cluster Computing (CLUSTER)*, 2012.

[17] J. Liu and B. Abali, "Virtualization polling engine (VPE): using dedicated CPU cores to accelerate I/O virtualization," in *Proceedings of the 23rd international conference on Supercomputing*, ser. ICS. New York, NY, USA: ACM, 2009, pp. 225–234.

[18] S. Kumar, H. Raj, K. Schwan, and I. Ganev, "Re-architecting VMMs for multicore systems: The sidecore approach," in *Proc. of the Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA) held in conjunction with the 34th Int. Symp. on Computer Architecture (ISCA)*, 2007.