Elektrotechnik & Informationstechnik (2018) 135/4-5: 304–308. https://doi.org/10.1007/s00502-018-0623-6

Smart mobility of the future – a challenge for embedded automotive systems

M. Baunach, R. Martins Gomes, M. Malenko, F. Mauroner, L. Batista Ribeiro, T. Scheipel

Smart, connected, and automated vehicles will have a significant impact on the safety, efficiency, and convenience of future transportation and mobility. However, most of the related services and technological features will be implemented in millions of lines of code running on hundreds of computers, embedded into each car. While classic automotive hardware and software are mainly designed statically and with just safety and real-time capability in mind, future systems also have to consider security and flexible maintenance aspects: Wireless communication across car boundaries requires solid protection against attackers and dynamic update mechanisms are required to reflect changing customer requirements and legal regulations throughout the entire lifetime of the cars. This article discusses specific challenges on embedded operating systems and processor architectures for highly dependable and compositional computing platforms in future vehicles.

Keywords: embedded automotive systems; smart vehicles; dynamic composition; operating systems; processor architectures

Mobilität der Zukunft – eine Herausforderung für eingebettete Systeme in Fahrzeugen.

Intelligente, vernetzte und automatisierte Fahrzeuge werden erhebliche Auswirkungen auf die Sicherheit, Effizienz und den Komfort zukünftiger Verkehrsmittel und unsere Mobilität im Allgemeinen haben. Die meisten zugehörigen Dienste und Funktionen werden jedoch in Millionen von Codezeilen implementiert, die auf Hunderten von Computern ausgeführt werden, die in jedes Fahrzeug sowie die Infrastruktur eingebettet sind. Während klassische Fahrzeughardware und -software hauptsächlich statisch und mit Blick auf funktionale Sicherheit und Echtzeitfähigkeit konzipiert ist, müssen zukünftige Systeme auch die Systemsicherheit und flexible Wartungsaspekte berücksichtigen: Drahtlose Kommunikationskanäle über Fahrzeuggrenzen hinweg erfordern einen soliden Schutz gegen Angreifer, und dynamische Aktualisierungsmechanismen sind erforderlich, um ständig wechselnde Kundenanforderungen und gesetzliche Vorschriften während der gesamten Nutzungsdauer der Fahrzeuge zu berücksichtigen. Dieser Artikel zeigt spezifische Herausforderungen an eingebettete Hardware und Software für hochzuverlässige und kompositorische Computerplattformen zukünftiger Fahrzeuge.

Schlüsselwörter: eingebettete automotive Systeme; Smart Vehicles; dynamische Komposition; Betriebssysteme; Prozessorarchitektur

Received March 19, 2018, accepted May 29, 2018, published online June 27, 2018
© The Author(s) 2018

1. Introduction

Smart Mobility is considered one of the 10 most disruptive technologies with a revolutionary impact on our lives, our society and our economy throughout the coming decades [10, 20]. One reason for this is that future vehicles, both individually and in conjunction with each other, as well as in close interaction with the infrastructure, will perform more and more tasks independently, which until now have to be accomplished by human drivers. This automation in the transport sector is expected to increase comfort and traffic safety ("zero accidents"), optimize traffic and logistical processes, and reduce energy consumption and environmental impact. It therefore interacts massively with technological advances in the equally future-oriented fields of robotics, artificial intelligence, manufacturing (Industry 4.0), sustainability and global connectivity (Internet of Things). It also introduces far-reaching consequences for the technologies in the vehicles. While mechanical and drive improvements alone are barely bringing significant progress, 90% of automotive innovation is already attributed to electronics or software and about 40% of the development costs relate to the associated embedded systems [7]. While this trend will continue, developing dependable and future-proof embedded automotive systems (e.g., Electronic Control Units, ECU) is becoming more and more complex due to increasing demands as shown in Fig. 1.

In-vehicle features Future vehicles will act autonomously. Therefore, Advanced Driver Assistance Systems (ADAS) and artificial intelligence (AI) will first process large amounts of sensor data to capture rapidly changing situations, identify threats, and make decisions about actions to be taken. The latter must in turn be carried out with high reactivity and by means of a large number of actuators. However, future data processing, learning and control algorithms will require far more software than before and far more computing power than currently available. Since increasing the computational power by adding more ECUs would increase the overall cost, weight, and power consumption, significantly more powerful com-

Baunach, Marcel, Institute of Technical Informatics, Graz University of Technology, Inffeldgasse 16, 8010 Graz, Austria (E-mail: baunach@tugraz.at); **Martins Gomes, Renata,** Institute of Technical Informatics, Graz University of Technology, Inffeldgasse 16, 8010 Graz, Austria (E-mail: renata.gomes@tugraz.at); **Malenko, Maja,** Institute of Technical Informatics, Graz University of Technology, Inffeldgasse 16, 8010 Graz, Austria (E-mail: malenko@tugraz.at); **Mauroner, Fabian,** Institute of Technical Informatics, Graz University of Technology, Inffeldgasse 16, 8010 Graz, Austria (E-mail: mauroner@tugraz.at); **Batista Ribeiro, Leandro,** Institute of Technical Informatics, Graz University of Technology, Inffeldgasse 16, 8010 Graz, Austria (E-mail: lbatistaribeiro@tugraz.at); **Scheipel, Tobias,** Institute of Technical Informatics, Graz University of Technology, Inffeldgasse 16, 8010 Graz, Austria (E-mail: tobias.scheipel@tugraz.at)

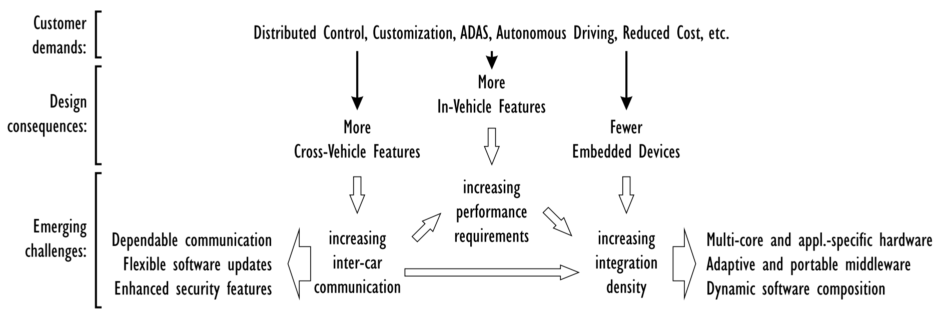

Customer demands:
Distributed Control, Customization, ADAS, Autonomous Driving, Reduced Cost, etc.
Design consequences:
More Cross-Vehicle Features
More In-Vehicle Features
Fewer Embedded Devices
Emerging challenges:
Dependable communication
Flexible software updates
Enhanced security features
increasing inter-car communication
increasing performance requirements
increasing integration density
Multi-core and appl.-specific hardware
Adaptive and portable middleware
Dynamic software composition

Fig. 1. Demands on smart mobility and emerging challenges for embedded automotive systems

puting platforms need to be developed to enable the consolidation of more software functions on fewer ECUs. By 2020, estimates assume that there will be around 100 million lines of program code [12] running on approximately 400 cores per vehicle. Driving the development of universal multi-core processors is only the first step. Eventually, highly application-specific hardware architectures will be used as well. This hardware diversity in combination with the increasing integration density and flexible customization will further increase the complexity of the software and require completely new concepts with regard to portability, modularity and dynamic composition as well as the associated variety of variants. This, in turn, has severe implications for the design of highly adaptive basic software and operating systems, which act as a link between hardware and application.

Cross-vehicle features Future vehicles will interact permanently. Distributed algorithms (e.g., for the large-scale optimization of traffic flows, the avoidance of accidents or the selective reduction of pollution in cities) require highly dependable wireless communication for intensive data exchange [3]. It is estimated that as many as 250 million networked vehicles are expected by 2020, which in turn will be part of the Internet of Things with more than 250 billion devices [29]. While vehicles have barely interacted wirelessly so far, radio communication opens up completely new possibilities. However, it must also be ensured that the networked embedded systems remain compatible and interoperable over the long term. If vehicles are to participate actively in the network (i.e., benefit from it and contribute to it) over an estimated lifetime of at least 10 years, then regular updates of the software due to changes in legal regulations, improved algorithms or new communication protocols must be expected. In addition, wireless communication is far more susceptible to environmental perturbation than in-vehicle wired bus systems and provide new attack surfaces for hackers who can now cause damage remotely without direct access to the devices. Again, support for immediate over-the-air software updates is essential, but once more requires modular software design and dynamic composition at runtime. In addition, the already mentioned processor architectures and basic software have to be equipped with sophisticated security mechanisms in order to isolate safety-critical software functions from each other, but to also allow interaction if necessary.

Challenges In order to tackle the demands of future mobility, a significant number of highly complex, highly integrated and massively connected embedded automotive systems will be used in automated and autonomous vehicles. In order to be prepared for use cases with strict demands on dependability, these systems have to undergo a profound change from static and monolithic designs towards much more dynamic and compositional concepts with security, safety, real-time, and maintainability in mind. As shown in Fig. 1, the concepts must support advanced multi-core and application-specific processor architectures, protection against environmental perturbation and attacks, dynamic update mechanisms, and simplified software portability for long-term operation.

This article describes four concepts towards a highly dependable and compositional embedded computing platform for future automotive use cases. Section 2 shows the *mosart*MCU [16] multi-core processor architecture with inherent operating system awareness. Section 3 tackles security concepts within hardware and software of the mentioned architecture to prevent malicious attacks. Section 4 illustrates a concept for dynamic but dependable software composition at runtime and based on the *MCSmart*OS [11] operating system. Section 5 shows an approach for simplified porting of the mentioned OS to new computing platforms. The final Sect. 6 concludes our article and summarizes our approaches.

2. Processor architecture: OS-awareness in multi-core MCUs

Most concepts in today's processor architectures date back to the 1970s until 90s. In particular, those architectures do not have inherent OS-awareness to support specific operating system features for dependable execution. The *mosart*MCU tackles this lack by integrating OS-awareness into a microcontroller unit: Being aware of the internal OS data structures and their placement in memory, our MCU is able to read and modify them concurrently to the kernel for improved real-time scheduling and security aspects. For instance, the MCU is always aware of the currently running task's priority and permissions as provided by the kernel, and might adjust various settings in the task control blocks.

We also proposed StackMMU [17], an extension which reduces the overall required task stack memory by addressing it virtually. Address translations are stored within the Task Control Block (TCB) of each task, and StackMMU is able to extend and reduce the stack on-demand, remaining completely transparent to the tasks. With CoStack [18] we extended StackMMU by a collaborative approach: If a high priority task cannot be served due to a less important task, the lower prioritized task is asked to voluntarily free stack memory. For real-time operation, both StackMMU and CoStack are designed to execute in predictable time. To also counteract unpredictable interrupt handling instants, we proposed EventIRQ [16]. Here, all the interrupt service routines (ISRs) are mapped to tasks and all the interrupt request (IRQs) are mapped to OS events. EventIRQ handles each event in hardware first and avoids unnecessary preemption of the currently running task unless the task waiting for the event has higher priority. The benefit of the mentioned concepts can be measured with the built-in performance monitoring unit [25].

Besides application-specific processor extensions, novel multi-core architectures also integrate more and more execution units on a system on chip (SoC). To increase the computational power, these cores can run software in parallel. However, interaction between code on different cores requires sophisticated synchronization mechanisms. Therefore, our SoC bus [19] supports remote instruction calls (RICs) to remotely execute instructions on another core. RIC respects task priorities and also supports the remote invocation of OS-aware functions (e.g., related to EventIRQ). In addition to multi-core ASICs, the availability of reconfigurable logic (field programmable gate arrays, FPGA) in embedded systems will cause a paradigm shift from static hard-wired hardware systems to highly flexible processors for adaptive and application-specific logic. Especially as soon as prices drop and become acceptable for mass production, this will enable new use cases but at the same time raise new challenges on hardware/software co-design.

3. Security: isolation and interaction

One of the main concepts in Smart Cities is having vehicles communicating between each other (V2V) and with the infrastructure (V2I). While this connected traffic scenario is built for efficiency and safety purposes, at the same time is becoming a very attractive target for cyberattacks. Vehicles today can be seen as yet another Internet device having the vulnerabilities known to conventional IT devices.

Prior to making vehicles connected, when data was exchanged only through internal and wired networks (considered trusted), future developments will extremely push the demand for flexible security mechanisms in embedded automotive systems. Today, with the introduction of several communication types inside the vehicle, especially V2IoT, the attack vector surface broadens. Remote attacks can hijack vehicle communication and data systems by stealing, modifying, or injecting maliciously crafted messages by hostile software providers, hackers, or even vehicle owners themselves. In autonomous cars, almost every car function can be accessed remotely, increasing the safety concerns. A lot of automotive cyberattacks have already been discovered [13, 21, 27]. They all use enhanced automotive features to directly access internal networks (e.g., remote diagnostics, mileage tracking system, cellular and Bluetooth links). By accessing the internal networks, they can take over control of every ECU and its functionality, including disabling breaks, stopping the engine, braking the wheels, etc. For this reason a secure automotive ECU architecture is needed.

A secure and cost-effective hardware base in coordination with minimalistic but secure kernel functions, needs to jointly provide strong software isolation, protection of sensitive data against malicious compromise, flexible sharing of peripherals, but with enhanced and controlled access at the same time, as well as isolated and authenticated inter-process communication (IPC). Furthermore, in order to allow secure deployment of new software components or attestation of already deployed software (to verify its authenticity), hardware-efficient cryptographic hash accelerators have to be implemented as well.

There are already several hardware-based security architectures designed in this direction [1, 4, 22], but we are proposing a solution based on an open-source hardware instruction set architecture (ISA) with various privilege levels (i.e., RISC-V [28]), on which a minimal microkernel is running (*MCSmart*OS). All memory accesses are mediated by a tailored memory protection unit (MPU) aware of code/mode-specific protection domains. Its role is to protect the system and the tasks from memory corruptions caused by faulty or malicious software components. Secure peripherals will be managed by secure drivers locking the I/O ports while in use, and interrupt service routines (ISR) will not be able to leak or access information they are not authorized to access. Communication between tasks will be handled through secure and authenticated IPC mechanisms, like secure message queues or secure shared memory. By delegating security functionalities to the hardware and properly designing minimal and secure system software that manages the applications running on the platform, a root of trust will be established. This will provide isolated task execution, but at the same time will allow secure communication between tasks and shared access to their peripherals.

4. Dynamic composition: pluggability and interoperability

A strong desire from the automotive industry is to enhance software maintainability, since, nowadays, every software modification demands a human supervised integration process, which is expensive, time consuming and error-prone.

To cope with this recurrent unfulfilled requirement, methods and techniques for the so called *dynamic composition* are necessary. *Dynamically composed systems* are able to perform partial updates on-the-fly and are able to decide whether or not a given change will keep the system still compliant with its dependability requirements. Key to achieving dynamic composition is *update authentication* and *compatibility analysis*. The compatibility of an update is assured by two properties: *pluggability* and *interoperability*. An update is pluggable if there is enough memory to store it and all dependencies with the proper versions are present in the system (libraries, resources, OS, etc.). An update is interoperable if it does not cause any timing or conflict issue at run-time [24] (deadlocks, starvation, deadline violation, etc.).

However, there is a price to pay for dynamic composition: memory and processing overheads. In the monolithic approach, all dependencies are checked at compile-time and the timing analysis is performed offline. In dynamically composed systems with a low number of software variants, it is still feasible to perform these checks offline, upon changes, for every variant. However, future systems tend to be highly customizable, which will lead to a huge number of variants, thus offline checks for every variant would be too expensive and time-consuming. Therefore, scalable solutions demand dependency checking and timing analysis to be performed at run-time, and for that, additional meta-data must be stored in the compiled software modules. Additionally, the systems must offer roll-back mechanisms, so that the devices remain functional even upon failures in the update process. It is worth to mention that the compatibility analysis demands high processing power and might not be feasible in resource constrained hardware. An option to avoid putting too much overhead onto the embedded systems is to develop update protocols that assign most of the load to servers (or to the cloud) [24].

The biggest challenge in achieving dynamic composition for embedded automotive systems is to develop an efficient interoperability check (scalable and lightweight) and to prove that the compatibility check will assure that the system remains dependable. These challenges are not easily solvable, but it is worth the effort, since such a dynamic system will be able to change even marketing and selling strategies. For instance, new cars can be sold with all the supported extras for a trial period, so that customers get used to a fully equipped car and end up buying some of the extras after the trial (all non-bought functionalities can be removed through a remote update). Another possibility is renting extras: one could wish adaptive cruise control only for a weekend trip, for example.

5. Middleware: operating systems and portability

The ubiquity of embedded systems in modern society presents many challenges to software and hardware developers. The number of devices in the upcoming Internet of Things (IoT), including autonomous and connected vehicles, is expected to increase exponentially [29], along with the diversity of those devices on both hardware and software side. Software developers, who currently focus on just a couple of different computing platforms, will be faced with a huge variety of devices, ranging from simple single-core to more complex multi-core or many-core systems, including specialized ASIC or reconfigurable FPGA components.

With such a diverse environment, multi-platform software must be easily portable, while maintaining its original dependability. Currently, AUTOSAR [2] provides automotive manufacturers and suppliers a common application interface and defines the basic software modules, improving collaboration between different parties on the software development and allowing the execution of any compatible application software independently from the underlying hardware.

Middleware portability, however, refers to adapting the basic software, including the operating system, to different or changing hardware architectures, e.g., different processors, reconfigurable logic or replaceable peripherals. Keeping the functional and non-functional behavior towards the application layer constant is a tough challenge. Retaining safety, security, and real-time behavior for highly critical systems (e.g., power plants, medical devices, autonomous vehicles) mostly requires manual coding and optimization within kernel code and data structures. For instance, the Linux [15] and FreeRTOS [8] kernels are available for a number of different architectures [6, 9, 14, 26], but porting requires a deep understanding of both the OS kernel and the target architecture. Since these and most other OSs are not optimized for portability, each port involves enormous implementation and testing efforts, and is still error-prone [5, 23].

*MCSmart*OS is designed for portability to virtually any embedded platform, and provides a dependable, yet dynamic software platform for embedded systems. Its portability is based on a formal specification of the kernel API and functionality and the relevant microcontroller (MCU) properties and instruction set architecture. From these API/ISA models, a generator can produce kernel code for different architectures, reducing the amount of manual coding and the likeliness for errors. This even allows kernel development totally independent from prospect target architectures, making it easier to (1) test new kernel concepts during OS development and (2) maintain the OS after deployment, even on changing hardware. Apart, the formal approach also allows formal proofs regarding the correctness and consistency of the OS and MCU models, leading to improved safety for both simple and complex ECUs.

6. Conclusion

Embedded automotive systems for smart mobility impose severe challenges on a vast range of features. This article has discussed various aspects and approaches for future system design in the automotive domain, including compositional software development and multi-core processor architectures with application-specific extensions. Therefore, we introduced our processor architecture *mosart*MCU as soft core for reconfigurable logic. Furthermore, we illustrated how security issues can be handled within hardware or software to prevent cyberattacks on e.g. the car's internal network. Our proposed dependable and flexible software stack can be dynamically updated at runtime and its middleware *MCSmart*OS can easily be ported to different hardware platforms, as its model-based design relies on formal specification.

Acknowledgements

Open access funding provided by Graz University of Technology. The endowed professorship and working group "Embedded Automotive Systems" is partially funded by AVL List GmbH, the Austrian Federal Ministry of Sciences, Research and Economy (bmwfw), and Graz University of Technology. Parts of the work were also performed within the LEAD-Project "Dependable Internet of Things in Adverse Environments", funded by Graz University of Technology.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

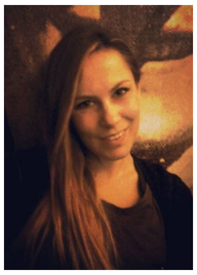
1. ARM Limited (2009): Building a secure system using TrustZone technology. Tech. rep., ARM.
2. AUTOSAR (2018): AUTOSAR: AUTomotive Open System ARchitecture. https://www.autosar.org/.
3. Boano, C., Römer, K., Bloem, R., Witrisal, K., Baunach, M., Horn, M. (2016): Dependability for the Internet of things: from dependable networking in harsh environments to a holistic view on dependability. E&I, Elektrotech. Inf.tech., 133(7), 304–309.
4. Brasser, F., El Mahjoub, B., Sadeghi, A. R., Wachsmann, C., Koeberl, P. (2015): TyTAN: tiny trust anchor for tiny devices. In Proceedings of the 52nd annual design automation conference, DAC'15 (pp. 34:1–34:6). New York: ACM.
5. Chou, A., Yang, J., Chelf, B., Hallem, S., Engler, D. (2001): An empirical study of operating systems errors. SIGOPS Oper. Syst. Rev., 35(5), 73–88.
6. Dall, C., Nieh, J. (2010): KVM for ARM. In Proceedings of the Linux symposium (pp. 45–56).
7. EY Global Automotive & Transportation Sector (2016): Automotive change drivers for the next decade. https://webforms.ey.com/Publication/vwLUAssets/EY-automotive-change-drivers-for-the-next-decade/$FILE/EY-automotive-change-drivers-for-the-next-decade.pdf.
8. FreeRTOS (2018): The FreeRTOS kernel. https://freertos.org.html.
9. FreeRTOS (2018): Official FreeRTOS ports. https://freertos.org/RTOS_ports.html.
10. Gartner Inc. (2015): Top 10 strategic technology trends for 2015. http://www.gartner.com/technology/research/top-10-technology-trends/.
11. Gomes, R. M., Baunach, M., Malenko, M., Ribeiro, L. B., Mauroner, F. (2017): A co-designed RTOS and MCU concept for dynamically composed embedded systems. In OSPERT'17.
12. IEEE Spectrum (2018): This car runs on code. https://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.
13. Isaac, J. T., Zeadally, S., Camara, J. S. (2010): Security attacks and solutions for vehicular ad hoc networks. IET Commun., 4(7), 894–903. https://doi.org/10.1049/iet-com.2009.0191.
14. Kleen, A. (2001): Porting Linux to x86-64. In Proceedings of the Linux symposium.
15. Linux Kernel Organization, Inc. (2018): The Linux kernel archives. https://www.kernel.org/.
16. Mauroner, F., Baunach, M. (2017): EventIRQ: an event based and priority aware IRQ handling for multi-tasking environments. In Proc. of the 20th euromicro conference on digital system design (DSD) (pp. 102–110).
17. Mauroner, F., Baunach, M. (2017): StackMMU: dynamic stack sharing for embedded systems. In Proc. of the 22nd IEEE int. conference on emerging technologies and factory automation (ETFA) (pp. 1–9).
18. Mauroner, F., Baunach, M. (2018): CoStack: collaborative stack sharing for embedded real-time systems. In Proc. of the 13th international conference on systems (ICONS) (pp. 32–37) ISBN 978-1-61208-626-2.
19. Mauroner, F., Baunach, M. (2018): Task priority aware SoC-bus for embedded systems. In Proc. of the 19th international conference on industrial technology (ICIT) (pp. 1453–1458).
20. McKinsey (2013): Disruptive technologies: advances that will transform life, business, and the global economy. http://www.mckinsey.com/insights/business_technology/disruptive_technologies.
21. Miller, C., Valasek, C. (2015): Remote exploitation of an unaltered passenger vehicle. Black Hat USA 2015.
22. Noorman, J., Agten, P., Daniels, W., Strackx, R., Herrewege, A. V., Huygens, C., Preneel, B., Verbauwhede, I., Piessens, F. (2013): Sancus: low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In Presented as part of the 22nd USENIX security symposium (USENIX security 13) (pp. 479–498). Washington: USENIX.
23. Ray, B., Kim, M., Person, S., Rungta, N. (2013): Detecting and characterizing semantic inconsistencies in ported code. In 2013 28th IEEE/ACM international conference on automated software engineering (ASE) (pp. 367–377).
24. Ribeiro, L. B., Baunach, M. (2017): Towards dynamically composed real-time embedded systems. In Logistik und Echtzeit (pp. 11–20). Berlin: Springer.
25. Scheipel, T., Mauroner, F., Baunach, M. (2017): System-aware performance monitoring unit for RISC-V architectures. In Proc. of the 20th euromicro conference on digital system design (DSD) (pp. 86–93).

26. Takata, H., Sugai, N., Yamamoto, H. (2003): Porting Linux to the M32R processor. In Proceedings of the Linux symposium (pp. 398–409).
27. Verdult, R., Garcia, F. D., Ege, B. (2013): Dismantling megamos crypto: wirelessly lock-picking a vehicle immobilizer. In 22nd USENIX security symposium (USENIX security 13), Washington: USENIX Association.
28. Waterman, A., Lee, Y., Avizienis, R., Patterson, D., Asanovic, K. (2016): The RISC-V instruction set manual. https://riscv.org/specifications/.
29. World Economic Forum (2015): Is this the future of the internet of things? https://www.weforum.org/agenda/2015/11/is-this-future-of-the-internet-of-things/.

Authors

Marcel Baunach
is professor for Embedded Automotive Systems and head of the corresponding working group at the Institute for Technical Informatics at Graz University of Technology, Austria. He received both his Diploma and Ph.D. with distinction from the University of Würzburg, Germany, where he also founded the research area Wireless Sensor/Actuator Networks with a focus on embedded systems, radio communication, and indoor localization. In 2011 he changed to the automotive industry as head of hardware development for automotive diagnostics. In 2013 he returned to academia and joined Graz University of Technology. His research area is compositional hardware/software/network-codesign with a strong focus on embedded multi-core architectures, real-time operating systems, and self-organizing wireless communication.

Renata Martins Gomes
received her engineer's degree in Computer Engineering at Universidade Federal de Itajubá, Brazil, and is currently a Ph.D. student at the Embedded Automotive Systems Group at the Institute for Technical Informatics, Graz University of Technology. Her research interests lie on portability of real-time operating systems, from the hardware and software models through code generation and verification.



Maja Malenko
is a Ph.D. student at the Embedded Automotive Systems Group at the Institute for Technical Informatics, Graz University of Technology. She received both her Bachelor's and Master's degree in Informatics and Computer Engineering from the Faculty of Electrical Engineering and Information Technologies in Skopje, Macedonia. Her research interest is embedded hardware and software design, with special focus on security in embedded devices.

Fabian Mauroner
received the B.Sc. degree in Computer Science from the University of Innsbruck, Austria, and the Dipl.-Ing. degree in telematics (renamed to information and computer engineering) from Graz University of Technology. He is currently a Ph.D. student at Graz University of Technology. His research interests include computer architecture up to operating system for embedded multi-core systems. Currently, he is working in the *mosart*MCU project, which implements operating system awareness into microcontrollers.

Leandro Batista Ribeiro
received the Engineer's degree (Dipl.-Ing.) in Computer Engineering at Federal University of Itajubá, Brazil. He is currently a Ph.D. student at the Embedded Automotive Systems Group at the Institute of Technical Informatics, Graz University of Technology. His interests are real-time, schedulability analysis and security aspects in embedded systems, and he currently works on design and implementation of dynamically composed embedded systems.

Tobias Scheipel
is a Ph.D. student at the Embedded Automotive Systems Group at the Institute of Technical Informatics, Graz University of Technology. He received both his Bachelor's and Master's degree (Dipl.-Ing.) in Information and Computer Engineering from Graz University of Technology. His research focuses on prototyping for embedded automotive multi-core systems with special interests in real-time operating systems and hardware/software codesign. He also works in cooperation with the industry in the same field.