A hypervisor-based fault recovery strategy for heterogeneous automotive real-time systems

Johannes Lex, Ralph Mader, Vitesco Technologies Group AG Regensburg
Prof. Dr. Ulrich Margull, Technische Hochschule Ingolstadt (THI)
Prof. Dr. Dietmar Fey, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Service-oriented software is becoming increasingly important in various cyber-physical systems which were in the past solely built with the focus on safety critical real-time tasks. The resulting systems are often heterogeneous in terms of hardware architecture, software architecture and criticality. This project targets on a fault recovery approach for such heterogenous real-time critical systems in the automotive area using hypervisor techniques.

Evolution of control units in the automotive industry

In the past, the software of automotive vehicles was mainly designed to execute safety critical real-time tasks. With the trend of autonomous driving and the Internet of Things, the demand for service-oriented software comes to this world at a fast pace. This leads to fundamentally different software- as well as hardware-architectures working together in one car. At the same time, the demand for fault tolerant software increases as the human driver is no longer available as backup in case of a fully autonomous driving car. To meet the demands for fault tolerance in such a heterogeneous system, a new fail-operational strategy has been investigated.

State of the art of fault recovery strategies

A classical strategy to increase the availability of critical parts is to multiply the hardware for each critical device. This is cost intensive and does not cover common cause failures in case the same hardware and software is used. Typical Electric-/Electronic-architectures for upcoming automotive systems consist of two fundamentally different types of Electronic Control Units (ECUs). On the one side, there are ECUs which are developed to execute real-time critical software on top of a microcontroller (µC) based hardware. Those ECUs will be referred to as RT-ECUs in the following. The leading software architecture for such RT-ECUs is the AUTOSAR Classic platform, where tasks are executed in the context of a real-time operating system (RTOS). On the other side, there are ECUs designed for the execution of service-oriented applications, called SOA-ECUs. Contrary to the RT-ECUs, their hardware is based on microprocessor (µP) technology. Here, the leading architecture for service-oriented software in the automotive industry is the AUTOSAR Adaptive platform [1], which in turn is usually executed on top of a POSIX OS like Linux. A solution which makes use of this inherent design diversity and redundancy of such heterogeneous systems can prevent common cause failures and is resource- and cost-effective at the same time. As such systems typically execute applications with different criticality, the freedom of interference needs to be guaranteed. One way to achieve this is the usage of hypervisors.

Current hypervisor-based fault recovery strategies are based on virtual machine (VM) migration like in [2–6]. Here, virtual machines and their states are moved from one ECU to another. With the techniques described in [2, 6] this is even possible for heterogeneous multi-processor systems, which means that VMs can be transferred from one processor architecture like x86 to another processor architecture like ARM by Instruction Set Architecture emulation or semantic migration. However, this is not available for µC architectures which lack virtual memory support or sophisticated virtualization features like VM migration. The strategies stated in [2–5] can be applied for fault tolerance in case the „hardware failure still allows for saving and transfer of the state of the VM“ [4] and soft real-time constrains can be guaranteed.

To the best of our knowledge, no hypervisor-based fault tolerant strategy was investigated for a heterogeneous setup involving µCs and µPs [2–4, 6–8] . Furthermore, we intend to meet hard real-time criteria for safety critical tasks and to recover the critical software in any case of hardware fault.

Concept of a hypervisor-based fault recovery strategy

This article shows how a SOA-ECU can back up real-time critical software from a RT-ECU using the hypervisor technology. The following requirements were considered for the elaborated backup procedure:

- real-time ready reaction time
- support of µC and µP hardware platforms and diverse software architectures
- guaranteed freedom of interference for software with different criticality

The first requirement is obvious for a real-time system. If a real-time critical task cannot be recovered within a certain time frame, the task fails and can lead to severe threats for health and life. A real-time ready approach on fault recovery in a distributed automotive system was already discussed in [9].

Furthermore, the recovery strategy should be applied in a heterogeneous system. The recovered task needs to run on a µP whereas the original task was executed on a µC. A redesign and reimplementation of the task for the µP system might be required. However, this effort can be minimized if the RTOS and applied safety mechanisms (e.g., memory separation, stack protection, ...) used on the RT-ECU are compatible with the µP of the SOA-ECU. In any case, the software needs to be compiled separately for the µP target and will be binary incompatible with the µC software.

The heterogeneity does not just affect the different hardware and software architectures of the system, but also the different criticalities of the tasks executed in the system. When a safety critical real-time task fails, severe threats for health and life could be the result, whereas a failure of a non-critical task does not bear any threat for health or life. Thus, a failure in the non-critical software should in no case be able to affect the behavior of the critical task in a malicious way. The so-called freedom of interference must be guaranteed.

With those requirements in mind, the following hypervisor-based failover strategy was investigated. As depicted in Figure 1, the RT-ECU executes its safety critical task during the default state (state A) and the SOA-ECU executes its non-critical applications. In addition, the SOA-ECU holds a backup of the critical functionality in its ROM. In case

of a fault of any kind on the RT-ECU, the SOA-ECU starts the recovery process (state B). The hypervisor instantiates a new partition as reaction to the fault (step 1), loads the backup of the critical function and starts it up (step 2). Afterwards, the restored critical task runs side by side with the service-oriented applications of the SOA-ECU on the same µP.

The hypervisor takes care of the freedom from interference while at the same time all resources of the system are available to the service-oriented software during normal state (state A) and the resources for the backup are only occupied in the failover state (state B).

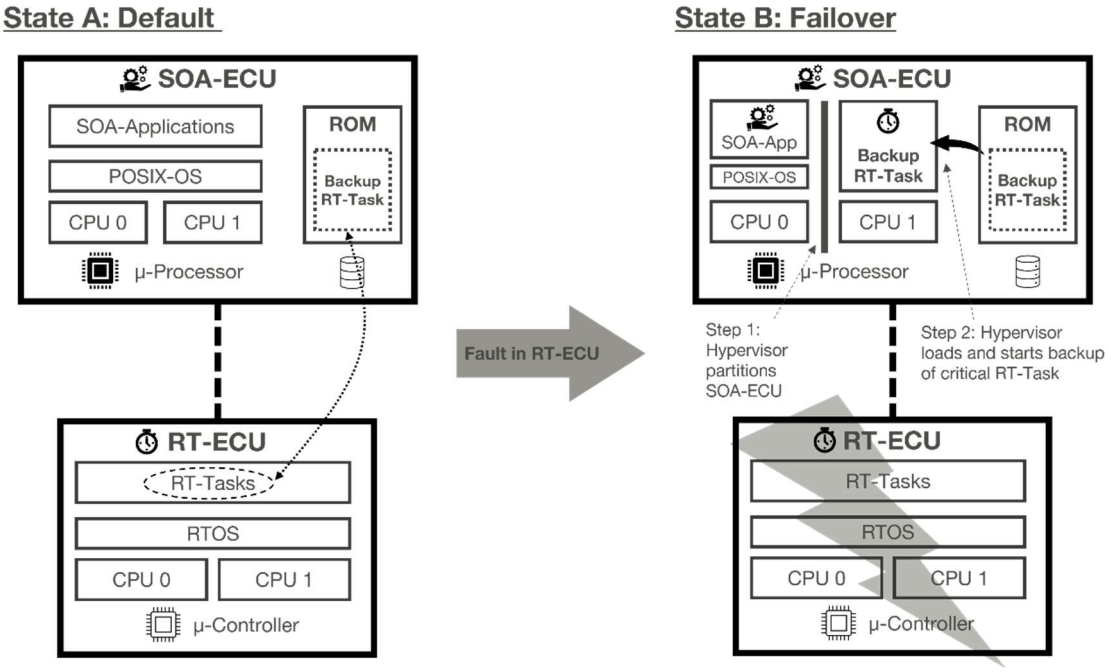


Figure 1: Transition from default state (A) to failover state (B). In case of a fault in the RT-ECU, the hypervisor creates a new partition on the SOA-ECU, loads the backup of the critical RT-Task in the new partition (step 1) and starts the new guest (step 2).

A stateless task was assumed in this setup as critical real-time task. In case this task depends on former states, replication methods like Input Backup Replication, Extended Primary Backup Replication or Extended Leader Follower Replication as described in [10] could be applied here. VM migration techniques like post-copy or pre-copy, which are well known from the server world cannot be applied here in the same manner, as A) the primary source might not be available during the migration process due to a hardware failure, B) such sophisticated hypervisor methods are not available on microcontrollers by today and C) the migrated data needs to be transformed for the varying CPU architecture like in [2, 6] with respect to the different handling of address spaces (virtual on µP and pure physical on µC).

Implementation and first measurements

For the implementation of this recovery approach, a hypervisor is needed, which can guarantee to not affect the real-time behavior of its guests, is able to create new guests during runtime "just in time" [6, 11] and, itself is not affected by a malicious behavior or even a crash of any of the remaining system. The Jailhouse hypervisor [12] was chosen which is a partitioning type 2 hypervisor and typically started from an running Linux.

After the startup and initialization phase, it runs bare metal and is not affected by faults or even a crash of the Host Linux system. The hypervisor partitions the system and assigns physical resources like CPU-cores or memory regions to its guests in a one-to-one manner without overcommitment of resources. Thus, the overhead introduced by the hypervisor is low and it has very little effect on the real-time capability to the executed backup task [13].

The time until a critical task is started up by the hypervisor is crucial in this setup. Since the execution of the jailhouse hypervisor only consumes very little resources, it can be started beforehand and executed permanently. Thus, the startup time of the Jailhouse hypervisor is therefore not considered in the following measurements. Instead, the creation time for a new partition and the startup time of a guest is considered solely.

For a first evaluation on the feasibility of this concept, this time was measured for the Jailhouse hypervisor on the Nvidia Jetson TX2 Board. This board is equipped with several General-Purpose Input/Output (GPIO) Pins, which were used to measure the startup time of a jailhouse guest, a so-called inmate. A new Jailhouse inmate was created which toggles a GPIO Pin. The Jailhouse Hypervisor is already activated by Linux before the tests start. During the testcase, the Linux system toggles a GPIO Pin from the default "low" state to "high". Afterwards, it instructs the Jailhouse Hypervisor to create a new partition and to load and start the inmate. The inmate takes over the control of the GPIOs and immediately pulls the GPIO Pin to "low" state again. In this way, the duration for the whole startup of a new inmate can be measured by the time span between the rising edge of the GPIO and its falling edge. The pin was traced by an oscilloscope and the resulting measurements for the startup time of the inmate were in the range of [109ms .. 115ms]. To the best of our knowledge, no detailed investigations on the startup time for jailhouse inmates was done with practical tests and no model for theoretical worst-case timings is available so far.

The startup time for a new inmate depends on various circumstances like the behavior of Linux or the size of the inmate as it needs to be loaded from the ROM into RAM before execution. Here, no RT-Patch was implemented for Linux and the used inmates were fairly small (22kB of size). The test revealed that the startup time of a guest can be in a reasonable time range for certain real-time critical automotive systems [9].

Conclusion and ongoing work

The depicted approach shows how a SOA-ECU can backup software from a RT-ECU in a resource efficient and safe way by using hypervisor technologies. Open topics concerning the elaborated approach include the development of state transfer or migration technique, further investigation of just in time instantiation [6, 11] for µC-/µP-based setups and a way to minimize the need for redevelopment of critical applications when porting them from the RT-ECU to the SOA-ECU.

Further development on the presented strategy is done on a test system for automotive ECUs consisting of a SOA-ECU executing an uncritical AUTOSAR Adaptive application and a RT-ECU executing a critical AUTOSAR Classic software. Both ECUs are connected via automotive ethernet and communicate via the SOME/IP Protocol. Current work is ongoing in enabling the Jailhouse Hypervisor and an AUTOSAR Classic software on the S32G µP from NXP.

References

[1] D. Reinhardt, U. Dannebaum, M. Scheffer, and M. Traub, “High Performance Processor Architecture for Automotive Large Scaled Integrated Systems within the European Processor Initiative Research Project,” in *SAE Technical Paper Series*, 2019.

[2] S. Groesbrink, “A First Step towards Real-Time Virtual Machine Migration in Heterogeneous Multi-Processor Systems,” *1st Joint International Symposium on System-Integrated Intelligence 2012: New Challenges for Product and Production Engineering*, vol. 2012.

[3] S. Groesbrink, “Virtual Machine Migration as a Fault Tolerance Technique for Embedded Real-Time Systems,” in *IEEE Eighth International Conference 2014*, pp. 7–12.

[4] A. Thind and M. Devgan, “Virtual Machine Migration Techniques used in Embedded Systems: A Review,” *IJCA*, vol. 144, no. 1, pp. 40–46, 2016, doi: 10.5120/ijca2016910099.

[5] S. Groesbrink, “Basics of virtual machine migration on heterogeneous architectures for self-optimizing mechatronic systems,” *Prod. Eng. Res. Devel.*, vol. 7, no. 1, pp. 69–79, 2013, doi: 10.1007/s11740-012-0421-7.

[6] P. Olivier, A. K. M. F. Mehrab, S. Lankes, M. L. Karaoui, R. Lyerly, and B. Ravindran, “HEXO,” in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, Phoenix AZ USA, 06172019, pp. 85–96.

[7] A. Burns and R. I. Davis, “A Survey of Research into Mixed Criticality Systems,” *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–37, 2018, doi: 10.1145/3131347.

[8] M. Cinque, D. Cotroneo, L. D. Simone, and S. Rosiello, “Virtualizing Mixed-Criticality Systems: A Survey on Industrial Trends and Issues,” *Future Generation Computer Systems*, vol. 129, no. 9, pp. 315–330, 2022, doi: 10.1016/j.future.2021.12.002.

[9] J. Lex, U. Margull, D. Fey, and R. Mader, “Fault Tolerance in Heterogeneous Automotive Real-time Systems,” in *Informatik aktuell, Echtzeit 2021*, H. Unger and M. Schaible, Eds., Wiesbaden: Springer Fachmedien Wiesbaden, 2022, pp. 73–82.

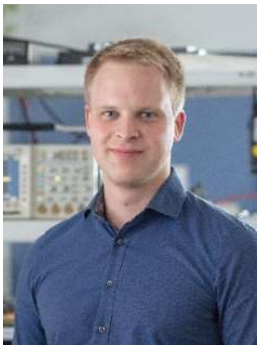
[10] T. Ishigooka, S. Honda, and H. Takada, “Cost-Effective Redundancy Approach for Fail-Operational Autonomous Driving System,” in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, Singapore, 2018, pp. 107–115.

[11] F. Manco *et al.,* “My VM is Lighter (and Safer) than your Container,” in *Proceedings of the 26th Symposium 2017*, pp. 218–233.

[12] *Jailhouse.* [Online]. Available: https://github.com/siemens/jailhouse

[13] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, “Look Mum, no VM Exits! (Almost),” *Proceedings of the 13th Workshop on Operating Systems Platforms for Embedded Real-Time Applications*.

Authors:



Johannes Lex is Software Architect at Vitesco Technologies in Regensburg and collaborates with the Technische Hochschule Ingolstadt and Friedrich-Alexander-Universität Erlangen-Nürnberg in terms of his doctoral thesis. His focus area is fault tolerant software for heterogeneous automotive systems. In this context, he deals with embedded Linux, RTOSes, hypervisors and the automotive software standards AUTOSAR Classic and Adaptive.

Dietmar Fey is a Full Professor of Computer Science with Friedrich-Alexander-University Erlangen-Nürnberg (FAU). After his study in computer science at FAU he received his doctorate with in 1992 also at FAU. He was an Associate Professor from 2001 to 2009 for Computer Engineering with University Jena. Since 2009 he leads the Chair for Computer Architecture at FAU. His research interests are in parallel computer architectures, memristive computing, and embedded systems.

Ulrich Margull is professor for Embedded Systems and Computer Engineering at the Technische Hochschule Ingolstadt (THI). His research activities are embedded and real-time systems in the automotive and avionics domain. In addition, he also gives lectures on quantum computing.

Mr. Ralph Mader studied Electrical Engineering at the University of Applied Sciences in Regensburg. Since 1991 he is working for Vitesco Technologies and its predecessor companies. He works there in the role of a Principal Expert for “Software Architecture Embedded Systems” and is heading the Software Advanced Team within Vitesco Technologies which focusses on the development of software technologies for master controller, e.g. for fail operational. Some of the concepts like AUTOSAR Classic Flexibility entered meanwhile the AUTOSAR Standard.