*d*OSEK: A Dependable RTOS for Automotive Applications

Martin Hoffmann, Christian Dietrich, Daniel Lohmann
Friedrich–Alexander University (FAU) Erlangen–Nuremberg, Germany
E-Mail: {hoffmann,dietrich,lohmann}@cs.fau.de

***Abstract*—Recent automotive systems exhibit an increased susceptibility against transient hardware faults. As a consequence, dependability measures are mandatory to provide appropriate fault detection or masking properties fulfilling the required safety standards. On the other hand, production costs are still a crucial factor in this domain, which leads to hardware consolidation and therefore mixed-criticality systems. An existing dependability approach, supporting such systems, combines triple modular redundancy with encoded operations, but still leaves the operating system as single point of failure. We intend to close this gap by extending the encoded operations throughout the kernel execution, and additionally integrate the analyzed system behavior into the code.**

I. Introduction

The effect of transient hardware faults is typically categorized into control flow and data errors. Accordingly, many software-based dependability concepts cover only one candidate of errors, for example introducing error-correcting codes for memory faults or control-flow signatures on the other hand. Only few approaches tackle both error types simultaneously, as for instance the vital coded microprocessor (VCP) [1], using an arithmetic data encoding also including control flow and temporal information. The VCP provides a common prime number A, allowing to detect errors when calculating the remainder, a variable-specific signature B_X preventing the mix-up of two encoded values, and a time stamp D ensuring actuality:

$$X_{enc} = X \cdot A + B_X + D \tag{1}$$

The concept was pursued by compilers generating fully encoded systems [2]. Nevertheless, such a fully encoded system implies considerable runtime overhead, which seriously restricts its applicability in our targeted domain of mixed-criticality systems. The *Combined Redundancy* (*CoRed*) approach [3] addresses this issue by safe-guarding the computation intensive application logic with triple modular redundancy, while protecting only the voting procedure – the remaining single point of failure (SPOF) – with the help an *Extended AN Code* (EAN), also based on Equation 1. While *CoRed* provides a holistic approach on application level, it still depends on an operating system, which reliably coordinates the replica execution, as well as spatial and temporal isolation. The hardening of the operating system itself is not covered by the *CoRed* approach, thus left as single point of failure. We intend to close this gap by extending the EAN concept throughout the entire kernel execution. Our targeted system domain are static OSEK-like operating systems (OSs), which are used extensively in the automotive area.

This work was partly supported by the German Research Foundation (DFG) priority program SPP 1500 under grant no. KA 3171/2-1, LO 1719/1-1 and SP 968/5-1.

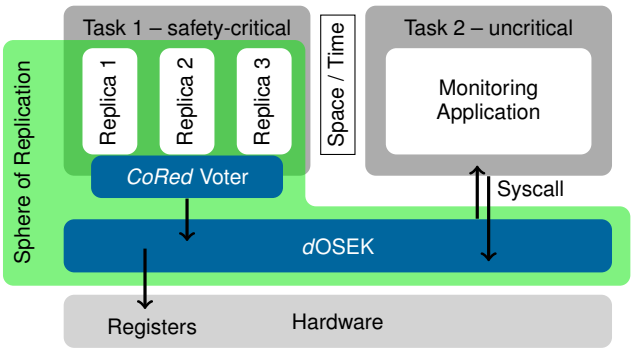

Fig. 1. Upon *d*OSEK two tasks are instantiated: Task 1 includes a *CoRed* [3] safe-guarded task system with triple modular redundancy. Task2 implements an uncritical monitoring application, which is isolated in terms of space and time from the safety critical component. *d*OSEK aims to extend the SOR from the *CoRed* application layer over the entire operating system.

II. The Basic Idea

We use the concept of the sphere of replication (SOR) [4] to depict the ideas and issues more comprehensively. A SOR describes a logical domain protected by some fault detection scheme ensuring that any fault occurring within the sphere and propagating to its boundary will be detected. In other words, as long as redundancy is applied, either being triple modularity, or in terms of encoded values, which facilitates at least fault detection, the system lies within a SOR. Once a value is decoded, the border of the SOR is reached: at this point the redundancy is dropped.

As depicted in Figure 1, our approach aims to extend the SOR from the *CoRed* application layer (replicas and voter) over the entire kernel execution, while preserving the realizability of mixed-criticality systems. The system call can be seen as an entry point to the kernel's SOR. At this point, redundancy in terms of EAN is applied and kept throughout the entire kernel execution. In fact, any data processed within the kernel, that is the input parameters as well as the internal kernel state, has to be encoded all the way through the kernel execution. Only when reaching the border of the SOR, for example when concrete values are to be set in hardware

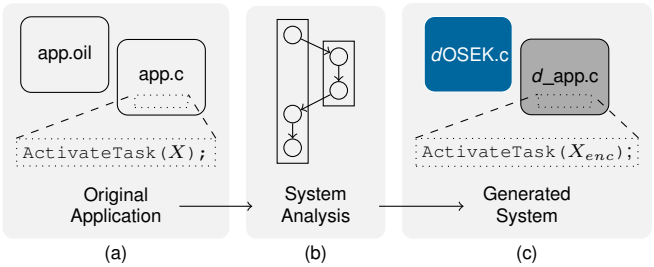

Fig. 2. On the basis of the original application (`app.c`) and the system description (`app.oil`) a system analysis reveals the task dependencies and the expected runtime behavior according to the OSEK specification. Subsequently a tailored kernel (`dOSEK.c`) is generated, where the expected behavior information is merged into *encoded* system call parameters.

registers, a value is validated and decoded. Consequently, all operations within the kernel must be able to handle EAN encoded values. This may also lead to unusual data structure design optimized for EAN support, rather than runtime or memory efficiency. However, a fully static embedded system provides a significant amount of system knowledge. To take advantage of this information, we aim to integrate the analyzed system behavior into the EAN code, which allows to detect deviations from the expected behavior.

III. Requirements

As already mentioned, our targeted system domain are static OSEK-like mixed-criticality systems. The OSEK specification [5] for the core functions is small, precise and a good starting point for designing a safety-critical OS. The *d*OSEK kernel will provide essential dependability aspects, like temporal isolation in terms of a watchdog or deadline monitoring concept. Also, spatial isolation among the tasks with the help of a memory protection unit is mandatory.

To realize *CoRed* protected applications, the *d*OSEK kernel has to provide means for reliable coordination and voting of the various replica tasks. Therefore the kernel must be tailored to the specific application to provide proxies for system calls within replicated instances. Such a tailoring requires a composable OS design concept similar to the CiAO family [6]. While CiAO concentrates on memory and runtime efficiency, we focus on effective dependability aspects, in the first place.

IV. Intended Approach

For both problems we will exploit the static domain knowledge, that arises from the static system design. As illustrated in Figure 2a, all tasks and resources are well-known beforehand. The stringent OSEK specification allows to build a complete space of reachable OS states. With the help of a static analysis tool [7], this, potentially huge, state space can be reduced by considering the analyzed kernel interaction and resource dependencies of the concrete application (see Fig. 2b). This, actually redundant, information can be then used as a further dependability aspect improving fault detectability: The signatures B of the *original* encoding rule (see Eq. 1) are selected to be unique for each variable, but they do not contain further redundant information. The system generation step (Fig. 2c) integrates the system knowledge into the code word, leveraging the validation of the tasks' interaction, as expected after the pre-runtime analysis. Entering the SOR with system call parameters encoded this way, the signature allows to check the validity of the call according to the analyzed application. As a result, beneath control flow and data integrity, even the system behavior is condensed into a tailored EAN code. The resulting kernel (`dOSEK.c`) has a strictly continuous SOR with dedicated entries and check points at the specific sphere boundaries, where the encoded results are validated.

This might further enable transactional behavior: When initiating a transaction, that is entering a system call, the current OS state can be backed up. The following kernel execution happens within the SOR; no irreversible action is performed without passing a check point at the SOR boundary. This clearly defined commit phase also facilitates graceful degradation. In case of a detected error. the misleading system call can be retried, or, with the help of the application knowledge, an alternative path can be chosen. Ultimately, if no viable solution can be found, the system may still result in a fail-stop state.

V. Challenges

We expect some interesting challenges to be tackled, when realizing the *d*OSEK kernel. First we need to find decent algorithms and data structures for the operating system primitives that operate on encoded values efficiently. Since the OSEK specification provides several conformance classes, we can start with a small subset of primitives and build up from there. The second challenge will be to keep the SOR as big as possible to minimize the unprotected operations. Clearly, the bare hardware registers are a natural SOR boundary of any software-based solution.

References

[1] P. Forin, "Vital coded microprocessor principles and application for various transit systems." in *Symp. on Control, Computers, Communication in Transportation (CCCT '89)*, Sep. 1989, pp. 79–84.

[2] U. Wappler and C. Fetzer, "Software encoded processing: Building dependable systems with commodity hardware," in *26th Int. Conf. on Comp. Safety, Reliability, and Security (SAFECOMP '07)*, F. Saglietti and N. Oster, Eds. Heidelberg, Germany: Springer, 2007, pp. 356–369.

[3] P. Ulbrich, M. Hoffmann, R. Kapitza, D. Lohmann, W. Schröder-Preikschat, and R. Schmid, "Eliminating single points of failure in software-based redundancy," in *9th Eur. Dep. Computing Conf. (EDCC '12)*. Washington, DC, USA: IEEE, May 2012, pp. 49–60.

[4] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proceedings of the 27th International Symposium on Computer Architecture (ISCA '00)*. New York, NY, USA: ACM Press, 2000, pp. 25–36.

[5] OSEK/VDX Group, "Operating system specification 2.2.3," OSEK/VDX Group, Tech. Rep., Feb. 2005, http://portal.osek-vdx.org/files/pdf/specs/os223.pdf, visited 2011-08-17.

[6] D. Lohmann, W. Hofer, W. Schröder-Preikschat, J. Streicher, and O. Spinczyk, "CiAO: An aspect-oriented operating-system family for resource-constrained embedded systems," in *2009 USENIX ATC*. Berkeley, CA, USA: USENIX, Jun. 2009, pp. 215–228. [Online]. Available: http://www.usenix.org/event/usenix09/tech/full_papers/lohmann/lohmann.pdf

[7] F. Scheler and W. Schröder-Preikschat, "The RTSC: Leveraging the migration from event-triggered to time-triggered systems," in *13th IEEE Int. Symp. on OO Real-Time Distributed Computing (ISORC '10)*. Washington, DC, USA: IEEE, May 2010, pp. 34–41.