Safety by Construction: Pattern-Based Application of Safety Mechanisms in XANDAR

Tobias Dörr*, Florian Schade*, Leonard Masing*, Jürgen Becker*, Georgios Keramidas†, Christos P. Antonopoulos†, Michail Mavropoulos†, Vasilios Kelefouras†, Nikolaos Voros†
*Karlsruhe Institute of Technology (KIT), Germany
Email: {tobias.doerr, juergen.becker}@kit.edu
†University of Peloponnese, Greece

***Abstract*—Considering the design of safety-critical embedded systems for future mobility solutions, the XANDAR project employs the X-by-Construction paradigm to meet non-functional requirements in an automated manner. This paper introduces the pattern library concept developed as part of the project and analyzes three state-of-the-art safety mechanisms for their compatibility with the approach.**

***Index Terms*—X-by-Construction, model-based design, multi-processor system-on-chip, embedded software, safety.**

I. Introduction

Embedded systems for technologies such as self-driving cars need to combine a high computational performance with the fulfillment of safety, security, and real-time requirements [1]. This turns the design of such systems into a complex task and creates the need for systematic approaches to tackle the associated challenges in a cost-efficient manner.

In the XANDAR project, a holistic toolchain providing such approaches is developed [2]. The strategy employed to achieve this is based on the X-by-Construction (XbC) paradigm, i.e., a "step-wise refinement process from specification to code" [3] that results in systems with guaranteed non-functional properties. The developed toolchain employs a model-based methodology to cover functional correctness as well as real-time, safety, and security requirements.

Focusing on multiprocessor system-on-chip (MPSoC) devices, this work presents first results and future directions of the XbC safety pattern concept in XANDAR. As part of this work, we introduce three selected safety patterns that we consider particularly interesting for the MPSoC-based design of safety-critical systems.

II. Process and Pattern Library

As shown in Fig. 1, the XANDAR development process can be divided into two overall portions: the model-based frontend and the XbC backend. In the frontend, users create a high-level description of the envisaged embedded system. During early development stages, the frontend is concerned only with platform-independent aspects at the software level. In this case, it allows the user to describe the intended software architecture as a network of Software Components (SWCs) interacting according to the Logical Execution Time (LET) paradigm [4]. Provided with a software architecture model and platform-independent code for each SWC, the toolchain

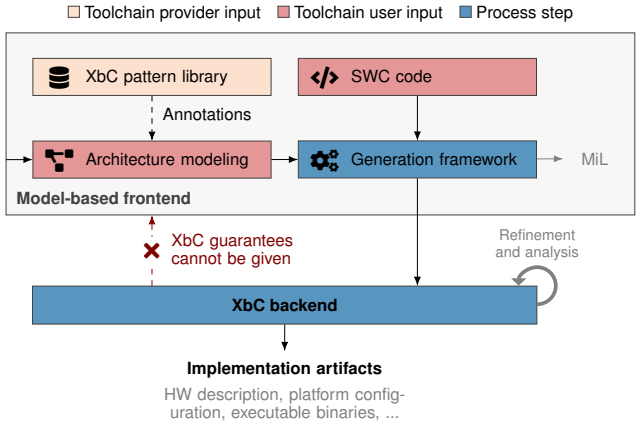

Fig. 1. High-level overview of the XANDAR development process

is able to perform a deterministic model-in-the-loop simulation (MiL) of the specified behavior. In later development stages, platform-specific aspects can be incorporated into the architecture model. Triggering the XbC backend initiates a refinement procedure that attempts to generate implementation artifacts with guaranteed runtime properties. If the backend is unable to provide the expected guarantees, it requests the user to revise the architecture model or the SWC code.

For the treatment of safety and security requirements, the toolchain employs a pattern-based approach. As visualized in Fig. 1, predefined *XbC patterns* are provided to the toolchain user in the form of an extensible library. Each pattern captures a specific safety or security mechanism, either known from the state of the art or specifically developed by XANDAR. In any case, a pattern consists of a verified design-time procedure and, optionally, trusted building blocks to be deployed to the target runtime. Note that some patterns (such as a pattern enforcing that two SWCs are mapped to distinct hardware units) do not generate active runtime entities. Such patterns do not trigger artifact generation, but their design-time steps still contribute to a safe and secure system architecture.

The toolchain user has the opportunity to annotate library patterns to the created software architecture model. As shown in Fig. 2, a software architecture contains an arbitrary number of SWCs, each with its LET parameters and an arbitrary number of input or output ports. Ports exhibit either sampling

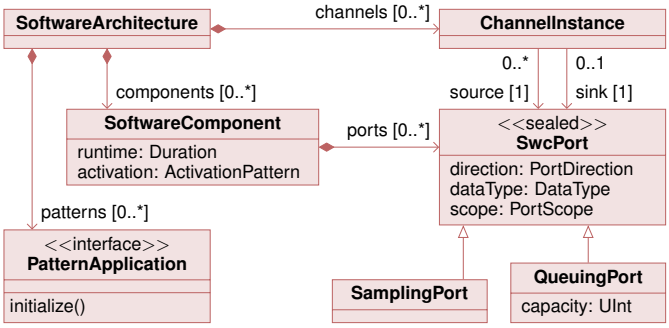

Fig. 2. Excerpt of the software architecture metamodel

or queuing behavior and can be connected to ports of opposite direction via channel instances. From the perspective of the toolchain itself, every pattern is provided as a class implementing the `PatternApplication` interface. The application of a pattern is therefore realized by the instantiation of the respective class. Using its `initialize()` method, a pattern has the opportunity to hook into the execution of both the generation framework and the XbC backend.

III. Safety Patterns for MPSoCs

Having described the pattern library concept of XANDAR, we now present ideas on how to integrate three selected safety mechanisms from an MPSoC context into it.

a) Hypervisor-based on-chip redundancy: An important target runtime of XANDAR are type-1 hypervisors. In such scenarios, SWCs mapped to a hypervisor partition can be replicated to implement fault masking based on spatial redundancy. A pattern realizing this idea is a specific Triple Modular Redundancy (TMR) scheme in which SWC copies as well as a voter are automatically generated and deployed to dedicated partitions. The resulting implementation is comparable to the "arbiter sandbox" approach presented in [5]. From a modeling point of view, an application of this pattern needs to reference only the SWC to protect. Since fault masking does not affect the nominal functionality of the system, the pattern has no impact on the simulation. Therefore, the XbC backend alone is able to generate the required runtime artifacts.

b) Cost-efficient fault tolerance: In cases where on-chip redundancy is not sufficient to obtain the necessary degree of reliability, but a full replication of an MPSoC is prohibitively expensive, the System-Level Simplex Architecture [6] is a promising concept to achieve fault tolerance. It is based on dynamic hardware redundancy and reacts to failures of one platform with the transition to a degraded functionality on another, more reliable platform. Due to the dynamic nature of this mechanism, provisioning it as a safety pattern is more challenging compared to the first case. An application of this pattern must reference two SWCs: one delivering the nominal and one delivering the degraded functionality. In addition, the fault detection procedure needs to be implemented by the user (as part of a SWC) and its result must be made available to the component orchestrating the pattern during runtime (via a SWC port). The behavioral aspect of this pattern has to be handled by the generation framework to make it visible in the simulation. The aforementioned runtime component to orchestrate the dynamic behavior must be highly dependable and generated by the XbC backend.

c) Hardware-enforced information flow control: To perform a logical isolation of physically connected on-chip components, modern MPSoCs are often equipped with Access Protection Units (APUs) attached to their internal interconnects. The work in [7] presents a model-based approach that uses such APUs to enforce an end-to-end information flow policy in a network of MPSoCs. This methodology is highly compatible with the XANDAR process and can be packaged as a safety pattern in a straightforward manner: A pattern application must contain a specification of all accepted information flows at SWC level, i.e., a list of ordered pairs where each pair declares a flow from a source SWC to a sink SWC as accepted. In the XbC backend, the pattern has to derive APU configurations from specified channel instances and, given these configurations, must ensure that all potentially feasible information flows are in line with the list of accepted flows, i.e., the desired information flow policy.

IV. Conclusion

In this work, we have given an overview of the XANDAR design methodology and its XbC pattern library. We were able to show that representative safety mechanisms from the state of the art are compatible with the library concept. While the software architecture metamodel is final, the pattern library is currently under development and will be populated with implementations of safety patterns such as the ones described above over the remaining course of the project.

Acknowledgment

This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957210.

References

[1] R. Ernst, "Automated Driving: The Cyber-Physical Perspective," *Computer*, vol. 51, no. 9, pp. 76–79, Sep. 2018.

[2] L. Masing, T. Dörr, F. Schade, J. Becker *et al.*, "XANDAR: Exploiting the X-by-Construction Paradigm in Model-based Development of Safety-critical Systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE '22)*, Mar. 2022.

[3] M. H. ter Beek, L. Cleophas, I. Schaefer, and B. W. Watson, "X-by-Construction," in *Leveraging Applications of Formal Methods, Verification and Validation: Modeling*, T. Margaria and B. Steffen, Eds. Springer International Publishing, 2018, vol. 11244, pp. 359–364.

[4] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Embedded Control Systems Development with Giotto," *SIGPLAN Not.*, vol. 36, no. 8, pp. 64–72, Aug. 2001.

[5] E. Missimer and R. West, "Distributed Real-Time Fault Tolerance on a Virtualized Multi-Core System," in *Operating Systems Platforms for Embedded Real-Time applications (OSPERT '14)*, Jul. 2014.

[6] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety," in *15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '09)*, Apr. 2009.

[7] T. Dörr, T. Sandmann, and J. Becker, "Model-based configuration of access protection units for multicore processors in embedded systems," *Microprocessors and Microsystems*, vol. 87, Nov. 2021.