

Parameterization of Fail-Operational Architectural Patterns

Dulcinea Penha, Gereon Weiss

Fraunhofer ESK

Hansastrasse 32, 80686 Munich

dulcinea.penha@esk.fraunhofer.de, gereon.weiss@esk.fraunhofer.de

ABSTRACT

In today's cyber physical systems, adaptability concepts can be used to fulfill fail-operational requirements while enabling optimized resource utilization. However, the applicability of such concepts highly depends on the support for the engineering during system development. We propose an approach to cope with the challenges of fail-operational behavior of CPS in which engineers are supported by design concepts for realizing safety, reliability, and adaptability requirements through the use of architectural patterns. The approach allows expressing concepts for fail-operational behavior at the software architecture level. By our approach, the effort for developing adaptive CPS can be kept low by utilizing fail-operational architectural patterns for general and reoccurring safety-relevant mechanisms. This is demonstrated by an application to an automotive case system.

Categories and Subject Descriptors

- Software and its engineering Domain-specific architectures

General Terms

Design, Reliability

Keywords

Software architecture, architectural patterns, cyber-physical systems, fail-operational requirements, safety, adaptability.

1. INTRODUCTION

Ensuring the safety and reliability of modern cyber-physical systems (CPS), even when failures occur, is essential and challenging for the development of these systems. To deal with such fail-operational requirements, engineers have to recur to concepts from the dependable systems area, like redundancy, monitoring, and special shutdown procedures. However, these techniques alone are not suitable for the application in certain domains. For example, in automotive systems redundancy usually adds prohibiting manufacturing costs due to additional hardware parts and tight integration of hardware and software.

In order to optimize the usage of resources in a source-scarce embedded system, one can take advantage of unused hardware

resources. This can be achieved either statically, by analyzing the system and defining scheduling and degradation strategies; or dynamically, by reconfiguring the activation or deployment of software systems at runtime. The applicability of such an approach highly depends on the support of the methodology to include adaptability. Current design approaches for CPS do not support the engineer in such a definition of adaptation concepts.

In this paper, we present an approach for patterns realization using models which integrates adaptability and fail-operational concerns into model-based development, in contrast to the classic definition of patterns assuming textual form (e.g., patterns catalogue [1]). Architectural patterns are thereby defined as meta-model extensions on the software architectural level, where the system-wide fail-operational requirements can be managed best. Thereby engineers are supported in assuring that safety requirements are considered at the architecture level.

2. RELATED WORK

In the following, we discuss previous work related to our approach of architectural patterns for fail-operational requirements in the context of safety-critical CPS. In literature, several authors propose catalogues of design and architectural patterns [1] [2], which partially consider safety and adaptation-oriented concepts. Pattern catalogues are usually provided as textual descriptions which, although very useful, do not integrate well into modern development approaches such as model-based development. Unlike these works, we do not attempt to provide a patterns catalogue; rather our approach takes advantage of patterns and integrates these into the context of safety-related CPS development.

One of the researched fault-tolerant patterns for redundancy is used exemplary to highlight our approach in Section 3. There are different fault-tolerant patterns which explicitly implement redundancy [1] [3] [4] [5] [6]. In order to support the design of safety-critical fail-operational CPS, our approach proposes to design redundancy supporting both homogeneous and heterogeneous copies. Besides, the number of versions is flexible and each version can be either active or passive, according to the system's specific fail-operational requirements.

3. FAIL-OPERATIONAL ARCHITECTURAL PATTERNS IN CYBER-PHYSICAL SYSTEMS DESIGN

The definition of patterns usually assumes a textual form and does not integrate well into modern development approaches. To take advantage of patterns and integrate these into the context of CPS software development, this work proposes the definition of patterns as meta-model extensions on architectural level. With our

approach, information concerning the recurring design as well as the involved fail-operational requirements is stored in patterns, at the same abstraction level of the software architecture. It allows associating the fail-operational-related strategies directly to the correspondent software components in the architecture, facilitating and automating its application.

3.1 Meta Patterns for Fail-operational CPS

This semi-formal representation of architectural patterns at the system's software architecture level enables the realization of patterns in the context of fail-operational systems, where several non-functional properties and requirements permeate the different abstraction levels of the design process.

In the automotive domain, for instance, the model-based design of systems involves specifying elements in EAST-ADL [7] and AUTOSAR [8] and includes stages such as feature modeling, functional architecture, design software and hardware architecture, implementation, code generation, and deployment. The knowledge transfer between these several different design phases is a challenge, especially if the information is not properly stored. Pattern descriptions, templates, and catalogues describe only informally the information, which is not integrated within the software architecture design and cannot be easily transferred to later phases of the development process. Furthermore, they do not assist the engineer in earlier and more abstract phases of the design process (e.g., when modeling the *Functional Design Architecture* in EAST-ADL).

In order to overcome this independent representation of design patterns, we introduce a modeling approach for architectural patterns. According to our approach, patterns are represented as meta-models which can be instantiated at the software architecture level. Patterns meta-models are created based on existent patterns descriptions with the necessary adjustments regarding fail-operational requirements. In order to fulfill these requirements, additional information is needed. For instance, for a hot-standby application this includes the number of hot-standby versions in the system, the *Fault Tolerant Time Interval (FTTI)* within which degradation has to occur, and several others. As part of the pattern (meta-model), they become formalized and can be instantiated for specific problems/applications. It enables the engineer to make use of arguments for safety and/or adaptability in the form of stored knowledge. The approach enables tracing of information, requirements, and mechanisms, helping to avoid failures during design.

Patterns are then implemented as models instances of these meta-models to meet the requisites of specific requirements in the software architecture in order to solve specific problems. The pattern model elements can directly reference software components in the architecture, enabling a tracing and coupling of the fail-operational requirements with the architecture. In the following a representative pattern with its fail-operational requirements has been exemplary chosen in order to evaluate the approach.

3.2 Patterns Parameterization

As discussed previously, when realizing a pattern several parameters are required in order to fulfill the pattern's purpose. This information is not necessarily listed in the pattern's common description. In the context of safety-critical systems several

safety-related and fail operational requirements must be parameterized: the maximum time interval in which a standby version has to take over in case of a failure, the number of redundant versions, if they are hot-, warm- or cold-standby, together with several other properties regarding fail-operational requirements.

In the following, safety- and adaptation-related requirements are discussed in more detail for the exemplary pattern-realization scenario, with respect to redundancy. Necessary adjustments have been introduced to this pattern in order to express adaptive and fail-operational aspects for designing CPS.

Modeling redundancy: There are different options for patterns to enable redundancy in the system. They contain either versions (*Heterogeneous Versions*) or copies (*Homogeneous Versions*) of the redundant function, in order to tolerate common-cause and/or different failures. In the context of current CPS, the redundant versions must not always be an independent functional equivalent software module which is implemented based on the same initial specification. For instance, in current driving-assistance systems, for cars which are not fully self-driving and the driver is still always available, an identical copy of the original software system is sufficient to implement standby techniques. Even a simplified redundant version of the software function might be sufficient in some cases. For instance, in case of a failure of the emergency break, the vehicle could just shut down the function and notify the driver. Moreover, safety-critical systems implemented and validated according to safety standards such as the ISO 26262 [9] should not present software bugs. In this scenario, not only the *Heterogeneous Redundancy* (different implementation), but also the *Homogeneous Redundancy* (equal implementation) pattern can be applied.

Furthermore, the number of versions or copies can vary according to the required safety and reliability levels. Additionally, the different versions can be active or passive as in the *Active-Active Redundancy* and *Active-Passive Redundancy* Patterns, implementing hot-, warm- and cold-standby. This can be influenced, for instance, by the maximum allowed take over time. In this way, the available individual redundancy patterns seem not to be the best choice for designing CPS. Hence, when realizing the redundancy pattern, the different configurations discussed above are considered, together with the necessary fail-operational requirements. The pattern is therefore called *adaptive fail-operational redundancy pattern* (AFOR Pattern) and is presented in the following.

Adaptive fail-operational redundancy (AFOR) pattern realization: As an example of the proposed approach, the AFOR pattern has been realized via meta-model extensions, as shown in Figure 1. Based on the aspects discussed above, the AFOR pattern is composed of N redundant versions and a voter. Versions can be either heterogeneous or homogeneous and are enhanced with a `StandbyClassification`, which can adopt the values hot-, warm- or cold-standby.

Regarding timing requirements with respect to failure recovery time, the AFOR pattern is enhanced with a `maxTakeOverTime`, meaning that the takeover process by a redundant version must happen within the given time interval. This value can be obtained from different analyses, for example from schedulability or worst-case-execution time analysis.

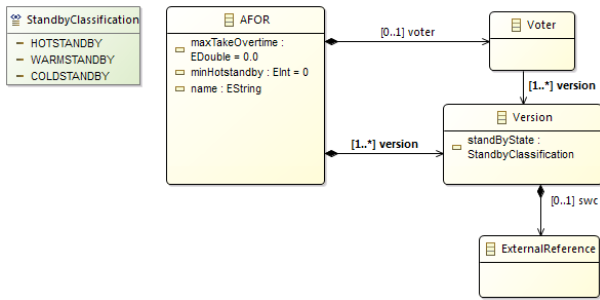


Figure 1. Meta-model for the adaptive fail-operational redundancy (AFOR) pattern

4. CASE-STUDY IN THE AUTOMOTIVE DOMAIN

In order to evaluate our approach, the AFOR pattern was applied to a representative case-study from the automotive domain: a simplified *Brake-By-Wire system (BBW)*. It is composed of a *BrakeTorqueCalculator*, a *BrakeController*, and two *ABSs (Anti-lock Brake-Systems)* components. On account of the high risks imposed by a failure of the BBW, a redundancy strategy is necessary. In this case study we focus on the ABS with an *FTTI* of 50 milliseconds and hot standby redundancy.

Our approach is applied to fulfill the specified fault-tolerant and safety requirements of the system. Hence, the meta-model presented in Figure 1 is instantiated for each of the ABS components. The AFOR pattern contains two hot-standby versions of ABS1 and the respective voter. Each version references the individual redundant component in the software architecture.

The employment of parameterized patterns on meta-model level enables reuse and adoption for different applications. Moreover, fail-operational patterns as model-instances become valuable assets in the model-based development process of CPS, since they enrich the architecture, are machine readable, enable the automatic generation of more detailed software architectures, as well as they allow traceability from requirements. The patterns can be used by engineers and tools to explicitly model aspects related to adaptation and fail-operational systems. Furthermore, the approach integrates well into modern model-driven development methods and allows machine readable specifications to be used for further analysis or engineering steps.

5. CONCLUSION

Supporting the engineers eases the cumbersome and error-prone manual task of fulfilling fail-operational requirements in novel software architectures of CPS. Our approach allows the explicit modeling of fail-operational related information and adaptive behavior at the architecture level. We have shown that fail-operational patterns employed as model-instances enrich the architecture. Moreover, they are machine readable, allow automatized approaches, and support traceability towards requirements.

In the future we want to extend the selection of fail-operational patterns for CPS. Moreover, we want to further investigate how we can propagate the information stored in the patterns fully automatically to later stages of the development, for example at runtime.

6. ACKNOWLEDGMENTS

A special thank you goes to those who contributed to this paper: Ludwig John for helping with the investigation of design and architectural patterns, during his internship, and Raphael Fonte Boa Trindade for his valuable comments.

The research leading to these results has received funding from the European Commission within the Seventh Framework Programme ([FP7/2007-2013] [FP7/2007-2011]) as part of the SafeAdapt project under grant agreement number 608945.

7. REFERENCES

- [1] Armoush, A. Design Patterns for Safety-Critical Embedded Systems. Ph.D. Thesis, Faculty of Mathematics, Computer Science and Natural Sciences, RWTH Aachen University, Aachen, Germany, 2010.
- [2] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Element of Reusable Object-Oriented Software. ISBN 0-201-63361-2. Addison-Wesley Professional, USA, 1994.
- [3] Avizienis, A. The N-version approach to fault-tolerant software. In IEEE Transactions on Software Engineering, Volume 11, No. 12, IEEE Press Piscataway, NJ, USA, 1985, 1491–1501.
- [4] Douglass, B. P. Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison-Wesley, New York, Boston, MA, USA, 2002.
- [5] Ahluwalia, K. S., Jain, A. High Availability Design Patterns. In Proceedings of the 2006 Conference on Pattern Languages of Programs (PLoP'06) (Portland Oregon, USA, 21-23 October, 2006). ACM New York, NY, USA, 2006, 24:1--24:11.
- [6] Grosspietsch, K. E. An adaptive approach for n-version systems. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'2003) (Nice, France, April 22-26, 2003). IEEE Computer Society, 2003, 215.1.
- [7] EAST-ADL - An Architecture Description Language for Automotive Software-Intensive Systems: <http://www.east-adl.info/> - last access December 2014.
- [8] AUTOSAR - AUTomotive Open System Architecture: <http://www.autosar.org/> - last access September 2014.
- [9] ISO 26262: Road vehicles – Functional safety (<https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-1:v1:en>) – last access December 2014.