Pattern-based methods for model-based safety-critical software architecture design: A PhD thesis proposal

Maged Khalil

Software and Systems Engineering
fortiss GmbH
Guerickestr. 25
80809 München
khalil@fortiss.org

Abstract: Software-intensive systems that perform safety-critical tasks are increasingly prevalent and pervasive in today's world. Driven by the incessant increase in the number of integrated control units, communication systems and software, managing architectural complexity, let alone mastering it, is becoming an increasingly difficult task.

Safety standards recommend (if not dictate) performing many analyses during the concept phase of development as well as the early adoption of multiple measures at the architectural design level, most of which has become part of the day-today business of safety-critical development, yet has to receive adequate tool support. This is particularly true if one wishes to front-load these aspects into an integrated solution environment, in which these (mostly) repetitive tasks can be automated. Model-based development techniques are increasingly used and well suited for these parameters.

Combining argumentation logic used for safety cases and safety concepts with abstract reasoning using model-based system description I argue about architectural optimization for safety-critical development. My approach allows reasoning about the system through the use of compositional description, which integrates physical environment models with system functional description models, and links problem-solving patterns with component model libraries which include nominal as well as faulty behavior.

1 Background and Motivation

1.1 Background

From advanced driver assistance and hybrid drives over autonomous high-speed rail and high precision medical equipment to Unmanned Aerial Vehicles: software-intensive systems that perform safety-critical tasks are increasingly prevalent and pervasive in today's world. Driven by the incessant increase in the number of integrated control units,

communication systems and software, managing architectural complexity, let alone mastering it, is becoming an increasingly difficult task. This difficulty in turn translates into a heightened possibility of design and implementation errors going undetected with hazardous consequences. This challenging situation is further exacerbated by the coupling of new development methods being employed for innovative technologies with increasingly complex international safety standards.

Non-functional requirements [La09] (the moniker under which safety is also bundled) should be dealt with from the beginning and throughout the software development process [CN00]. Ineffectively dealing with NFRs has led to a series of failures in software development [BLF99], [Li93]. Numerous literature examples have long existed [Br87], [CL99], pointing out these requirements as the most expensive and difficult to deal with. Functional requirements are naturally expressed in discrete, logic-based formalisms, which facilitate arguing about them at the architectural level. However, to express many non-functional requirements, software designers need to be able to integrate real-valued quantities representing physical constraints and probabilities into architectural description [HS06].

Safety standards recommend (if not dictate) performing many analyses during the concept phase of development as well as the early adoption of multiple measures at the architectural design level [EN08], [IEC09], [ISO11], [ARP11], [DO92], [DO12], most of which has become part of the day-to-day business of safety-critical development.

Because these analyses and architectural measures have to be conducted during the design phase of the systems, they cannot be based on physical prototypes and implemented software, and, hence, have to rely on the design information in terms of requirements, the structural design, functional specifications, and the principled knowledge about the (nominal and potential deviating) behavior of the components used [Bö11]. The complexity of such systems (regarding both structure and behavior) makes this a sophisticated task for experts and combined with the repetitive nature of the analyses carried out calls for computer-based automation and support. Any computer tool for this task that goes beyond editors and calculations and aims at supporting the core inferences, namely determining the consequences and risks implied by assumed faults based on design information and 1st principles knowledge, has to be knowledge-based, or, more specifically, model-based.

1.2 Related Work

Indeed, there exist results from previous work that form a good starting point for such an attempt. The field of model-based problem solving [St11] has generated theoretical foundations, prototypical solutions, and products for modeling artifacts and natural systems and for using models to solve a variety of tasks, with a major focus on automated diagnosis, but also solutions for failure modeling as well as for automated software FMEA [PS08], [SF12]. Several projects also exist which target the inclusion of reasoning about safety into the foundation of software development activities at the meta-modeling level in industry wide efforts, such as the AUTOSAR software reference architecture or EAST-ADL as well as the many research projects addressing these areas

(on the European scene alone, MAENAD, ATTEST2, TIMMO-2-USE, AMALTHEA, SAFE… etc.), which have shown progress but are far from a comprehensive solution.

On the other hand, there is extensive work on formalized reasoning about safety-cases and safety concept argumentation [PM99], [KE04], [KW04], which culminated in a standardization of the structured notation known as GSN [KH11]. An analysis of industrial safety-cases yielded a finite set of patterns [WS10] from which we plan to construct a pattern library of problem types and their respective solutions and use it to automate the generation of and argumentation about safety concepts as well as safety cases in our research CASE tool AutoFOCUS3 [AF3].

Several works exist on architectural benchmarking and design space exploration for architectural constraints. Works seeking to structure and facilitate the selection of architectural patterns and measures adequate for safety-critical requirements have already been presented [Ar10], but the selection process is still highly manual and case dependent.

1.3 Motivation

Yet there is no unified systematic approach to integrate safety-critical architectural constraints into the early phases of product development, before functions are divided unto logical or hardware components (such as all UML based approaches), while approaches that attempt to describe functionality free from component attachments are mostly limited to ADLs (e.g. feature models in EAST-ADL) and as such do not yet offer a seamless development capability. More specifically there is no approach that derives this integration from a solid argumentation about the selected solution pattern, its validity or the effects on software architecture correctness. Various approaches rooted in architectural design principles have tried to practically handle this discrepancy by recommending patterns which achieve many of the architectural constraints recommended by the standards, such as freedom from interference, usually by partitioning (separation of critical from non-critical software), and multiple redundancies (multiple version, dissimilar or independent software). Indeed, robust software architectures and efficient algorithms are still designed individually, not automatically generated, and this will likely remain so for some time. The emphasis, therefore, shifts from design synthesis to design verification—the proof of correctness. In summary, integrated models, methods and systems for fault analysis and risk assessment of cyber-physical systems are lacking, which is why I propose to develop coherent modeling formalisms and inference engines covering both physical and software systems, allowing the selection of and argumentation about architectural suitability and optimization and starting from the foundations stated above. Adequate tool support is central to the proposed approach, more so to deal with the highly repetitive nature of the complex tasks involved (e.g., FMEA), especially when dealing with variants, as most analyses have to repeated for every (even seemingly trivial) change, which would otherwise increase complexity and costs dramatically.

2 Solution Proposal

As previously mentioned, an approach based on compositional elements using which new or modified system variants can be (semi-)automatically (re-)configured and generated. More precisely; a model-based approach based on the consistent use of problem solving patterns and model element libraries.

2.1 Rationale

The proposed approach will be supported by a tool implementation based on an (deeper than hitherto achieved) integration of the physical system modeling and software component modeling which exploits the synergies between classical engineering, model-based problem solving and model-based software development paradigms. This approach is based on the following reasoning.

1- Hazard definition and avoidance is solely inferred by the interaction of the vehicle - as a physical system - with its physical environment (e.g., Vehicle under-braking results in an increased stopping distance which increases risk of collision.) A possible exception is an erroneous display of critical information to the driver, which leads to a critically wrong decision.

2- The behavioural model of the physical parts of the system and its interaction with the environment thus provides the primary inferences governing the requirements, analysis and function of the embedded software.

3- The embedded software thus only interacts with the physical system through this interface. Moreover, software errors do not directly lead to hazards in the environment per se, but rather must first lead to hazardous behaviour in the physical system by erroneous manipulation of the interface signals. As such, an incorrect vehicle speed display is not necessarily hazardous, while the severity of a false braking signal can only be determined by its impact on the physical system and its interaction with the environment. This view will govern the approach's handling of analysis for software components.

4- The functional and safety requirements for the embedded software are as such initially determined by the interaction model of the physical components with the environment. This model focuses the requirements and essential functions of the software: functional requirements are limited to the physically possible or relevant input and output combinations at the interface description, while the safety requirements are determined by the safety-critical scenarios and the physical actions leading to them.

2.2 Goals and Requirements

Thus a clear and concise understanding and representation of system functions, their relationships and dependencies as well as their their effect in the real world is essential. To manage complexity and because many (especially implementation) details are still unknown in early project phases; this representation must maintain a suitable level of abstraction, preferably at the functional network (functionality specification) level and before deployment to logical components. Furthermore, this representation must not only offer an abstract view of the system functionality during nominal component behaviour, but must support linking or deriving error models from them.

To master the complexity of current and future systems, the representation of complex operation modes and varying (partial-) functionality should be supported. This includes functional degradation and fall-back solutions..

By integrating hazard and risk analyses into the development environment, this approach provides a basis for the systematic and automated derivation of requirements, including relevance and completeness checks, for the safety concept and subsequent solutions as well as possibly enabling a synthesis of functional component description based on the physical models.

2.3 Summary

Underlying the approach is the consistent use of patterns from the categorization of hazard types, over the abstract modeling of the respective safety concepts, and down to their implementation in the system architecture description, with a focus on providing argument chains. The use of component model libraries that include with integrated behaviour descriptions for nominal and faulty conditions plays a central role. This could allow, for example, the automation of safety-critical optimized architecture pattern selection, based on the work in [Ar10] and in a later step, possibly a (semi-) automated architecture generation for each (sub-) system according to the selected safety concept.

This submission represents a work-in-progress, with several research project results still in various stages of completeness, and is meant to serve as a starting point for further discussion. A more detailed overview of the approach used and first results and prototypes are the subject of a future paper.

References

[AF3] AutoFOCUS 3, research CASE tool, af3.fortiss.org, 2012 fortiss GmbH

[ARP11] European Organisation for Civil Aviation Equipment, EUROCAE ED-79/ARP-4754: Certification considerations for highly integrated aircraft, 2011.

[Ar10] Ashraf Armoush, “Design Patterns for Safety-Critical Embedded Systems”, Ph.D. Thesis, RWTH-Aachen, 2010

[BLF99] K.K. Breitman, J.C.S.P. Leite, and A. Finkelstein, The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study, J. of the Brazilian Computer Soc., vol. 6, no. 1, pp. 13-38, July 1999.

[Bö11] „Funktionale Sicherheit: Grundzüge sicherheitstechnischer Systeme", Börcsök, J. / VDE-Verlag, 2011

[Br87] F.P. Brooks Jr., No Silver Bullet: Essences and Accidents of Software Engineering, Computer, no. 4, pp. 10-19, Apr. 1987.

[CL99] L.M. Cysneiros and J.C.S.P. Leite, Integrating Non-Functional Requirements into Data Model, Proc. Fourth Int'l Symp. Requirements Eng., June 1999.

[CNY00] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering. Kluwer Academic, 2000.

[DO92] European Organization for Civil Aviation Equipment (EUROCAE) and United States Department of Defense (DOD), EUROCAE ED-12B/DO-178B: Software considerations in airborne systems and equipment certification, 1992.

[DO12] United States Department of Defense (DOD), DO-178C: Software considerations in airborne systems and equipment certification, 2012.

[EN08] European Committee for ElectroTechnical Standardization, EN-50126 Railway Standard, www.cenelec.eu, 2008.

[GH08] Grunske, L. and Jun Han, A Comparative Study into Architecture-Based Safety Evaluation Methodologies Using AADL's Error Annex and Failure Propagation Models, HASE 2008. 11th IEEE High Assurance Systems Engineering Symposium, 2008.

[HS06] Thomas A. Henzinger, EPFL & Joseph Sifakis, Verimag, The Discipline of Embedded Systems Design, IEEE Computer Society 2007 – an update of The Embedded Systems Design Challenge, Proc. 14th Int'l Symp. Formal Methods, LNCS 4085, Springer, 2006, pp. 1-15.

[IEC09] International Electrotechnical Commission, IEC 61508 Standard, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems", www.iec.ch/functionalsafety, 2009.

[ISO11] International Standards Organization, ISO 26262 Standard, Road Vehicles Functional Safety, www.iso.org, 2011.

[Ke04] Tim Kelly, A Systematic Approach to Safety Case Management, SAE 2004 World Congress & Exhibition, March 2004, Detroit, MI, USA, Session: Safety-Critical Systems.

[KH11] Tim Kelly, Ibrahim Habli, et al.: Goal Structuring Notation (GSN). GSN COMMUNITY STANDARD VERSION 1, Origin Consulting (York) Limited, on behalf of the Contributors, November 2011

[KW04] Tim Kelly, Rob Weaver, The Goal Structuring Notation – A Safety Argument Notation, Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases.

[La09] Axel van Lamsweerde, Requirements Engineering - From System Goals to UML Models to Software Specifications, 2009.

[Li93] D.R. Lindstrom, Five Ways to Destroy a Development Project, IEEE Software, pp. 55-58, Sept. 1993.

[PM99] Papadopoulos Y., McDermid J.A., The Potential for a Generic Approach to Certification of Safety-Critical Systems in the Transportation Sector, Reliability Engineering and System Safety, 63(1):47-66, Elsevier Science,1999.

[PS08] Chris Price and Neal Snooke, An Automated Software FMEA, Proceedings of the International System Safety Regional Conference, Singapore, April 2008

[SF12] Struss, P., Fraracci, A.: Automated Model-based FMEA of a Braking System. In: 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (Safeprocess 2012), Mexico City, 2012

[St11] Struss, P. : A Conceptualization and General Architecture of Intelligent Decision Support Systems. MODSIM2011, 19th International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand, December 2011, pp. 2282-2288. ISBN: 978-0-9872143-1-7.

[WS10] Stefan Wagner, Bernhard Schätz, Stefan Puchner, Peter Kock, A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models, Proc. International Symposium on Software Reliability Engineering (ISSRE '10), IEEE Computer Society, 2010