Model Driven Development of Embedded Systems

Thomas Kuhn, Pablo Oliveira Antonino
Fraunhofer IESE

The model driven development is a proven technique for managing the complexity of modern embedded systems. Nevertheless, most MDD approaches targeting requirements specification and architecture models of embedded systems do not support the specific needs of embedded domains, preventing both the modeling of core concepts and the model based analysis of resulting design models. In this paper, we give an overview on the Fraunhofer Embedded Systems Modeling profile that adds specific views and modeling elements to the UML/SysML modeling languages to enable description of embedded systems specific properties.

Model driven development is an important technique to manage the complexity of modern embedded systems. It enables to address the challenges of system development in a more systematic way, which include multi-functionality, distributed execution, real-time properties, and safety-criticality [6]. For instance, domains such as automotive and avionics demand high integrity levels between hardware and software to ensure proper execution of their systems, which, in turn, are constantly becoming larger and more complex. A commercial airplane, for example, contains systems that control ground proximity, navigation, and engine commands, amongst others.

In this regard, some model-based approaches have been developed for supporting model-driven development of embedded systems, like for instance, the SPES 2020 methodology [1]. However, there is still a lack on means to properly support engineers during the modelling process. For example, the SPES 2020 approach provides generic methodological means that can be instantiated by different modelling languages like UML and SysML. Indeed, it is still necessary to provide means to properly support modelling of embedded systems that consider particular aspects of the domain such as hardware interfaces, processing units, and interruptions, among others.

For addressing this issue, this paper presents the Embedded Modelling Profile, which aims at equipping engineers with elements that are appropriate to model embedded systems characteristics.

Embedded Systems Architecture Modelling in Practice

The UML (Unified Modeling Language) is the most wide-spread general-purpose language for modelling software systems. It supports a diversity of elements and diagrams, which provide means for expressing systems from diverse perspectives [3]. Tailoring is possible with UML Profiles, which enable modeling particularities of domains by means of customizations of UML's syntax and semantics [5]. UML Profiles are mechanisms for specifying rules to be used in parts of the model of a system where specific constraints are required [7]. Profiles are based on stereotypes and tags, which are the concrete entities that must be applied to UML elements such as classes, components, and connectors, with the aim of ruling parts of a system with respect to constraints of the domain or of the modeling process. SysML is a dialect of UML that aims at supporting the specification, analysis, design and verification of systems and systems-of-systems that is realized as a UML profile. The reduced set of diagrams and elements

makes it easy to apply and learn when compared with the pure UML. Because of that, it has been used by practioners working in the development of embedded systems.

The Embedded Modelling Profile

In this section, we present the Embedded Modelling Profile, which is the core contribution of this paper. It closes the gap of UML and SysML with respect to views and modeling elements for specification of software intensive embedded systems by including best practices from the MDD and Software and Systems Architecture disciplines.

Models describing the systems structure should address concerns of the various stakeholders involved in the system specification and development. In this regard, we understand that it is important to have a clear definition of the drivers that motivate the system design to be as it is (cf. Figure 1).

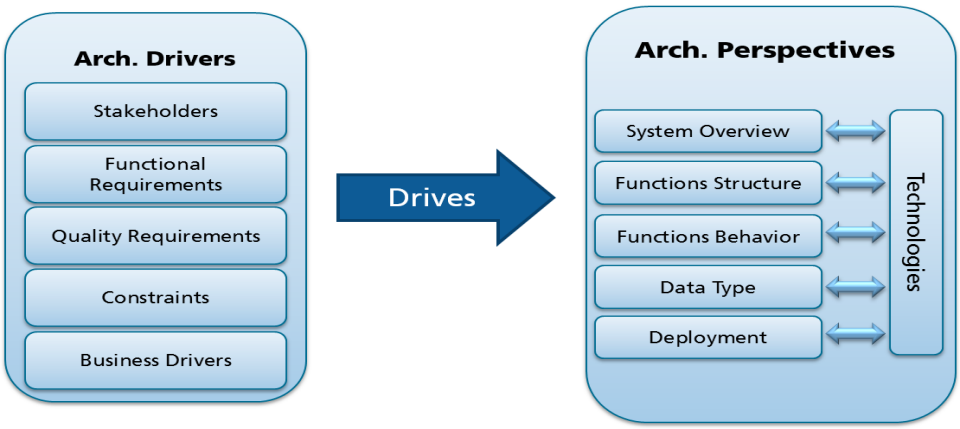

Figure 1. Architectural Drivers and Architecture Perspectives as main building blocks of architecture specifications.

Once the architectural drivers are defined, the architecture description should address the following compositional blocks of the system under development:

- **System overview / Context overview**: it describes the system as a black box, and points out how it communicates with externals entities. This is the highest level of abstraction of the architecture specification.
- **Function network:** Once these high level elements are identified, they should be refined in elements that represent system functions as well as their interactions. Later, these functional entities are then refined into elements that indicate how they are realized as software and/or hardware components.
- **Functional Behavior:** The functional behavior specification details the dynamics of interaction between functions. It is usually done using state charts and sequence diagrams.
- **Data Model:** It describes in detail the data that is exchanged between the system components.
- **Deployment:** It describes hardware entities that directly implement system functions, and also it describes the deployment strategy of logical components to hardware entities.
- **Technologies:** It is also important to indicate in the architecture specification which technologies will be used to implement the concepts described in the architecture views. Documenting technological decisions at this stage is an important activity because the technological constraints described in the architectural drivers can also be taken in consideration.

We have specified a modeling profile that supports the creation of architecture models, considering the architecture perspectives that are described in Figure 2. We have condensed this into three views: Functional View, Logical View, and Technical view. From now on, we will present the meta-model for modeling elements that comprise these views. Due to space constraints, the embedded systems modelling metamodel is not fully presented. However, we understand that the content presented in this paper provides enough information to support the understanding of the approach.

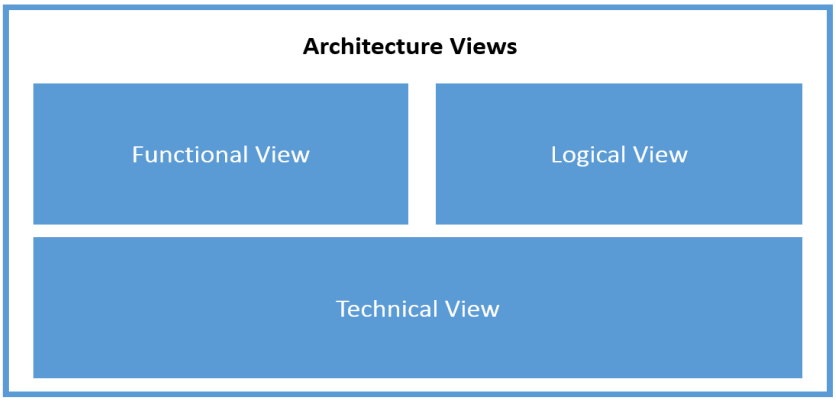

Figure 2. Architecture view that comprise the architecture perspectives aspects described in Figure 1.

Figure 3 shows the meta model for creating the Functional View. It enables the modeling of function nets that describe dependencies between system functions. Functions realize behavior of the system and exchange information. At this level, it is not specified how a function and this information exchange is realized. Function network models are used, for example, to analyze propagation of failures. Functions can be modeled individually as behavior functions and aggregated into function groups. Optionally, it is possible to refer to functions defined in function groups to improve readability of diagrams. From the modelling perspective, these elements are represented by UML class elements or SysML blocks.

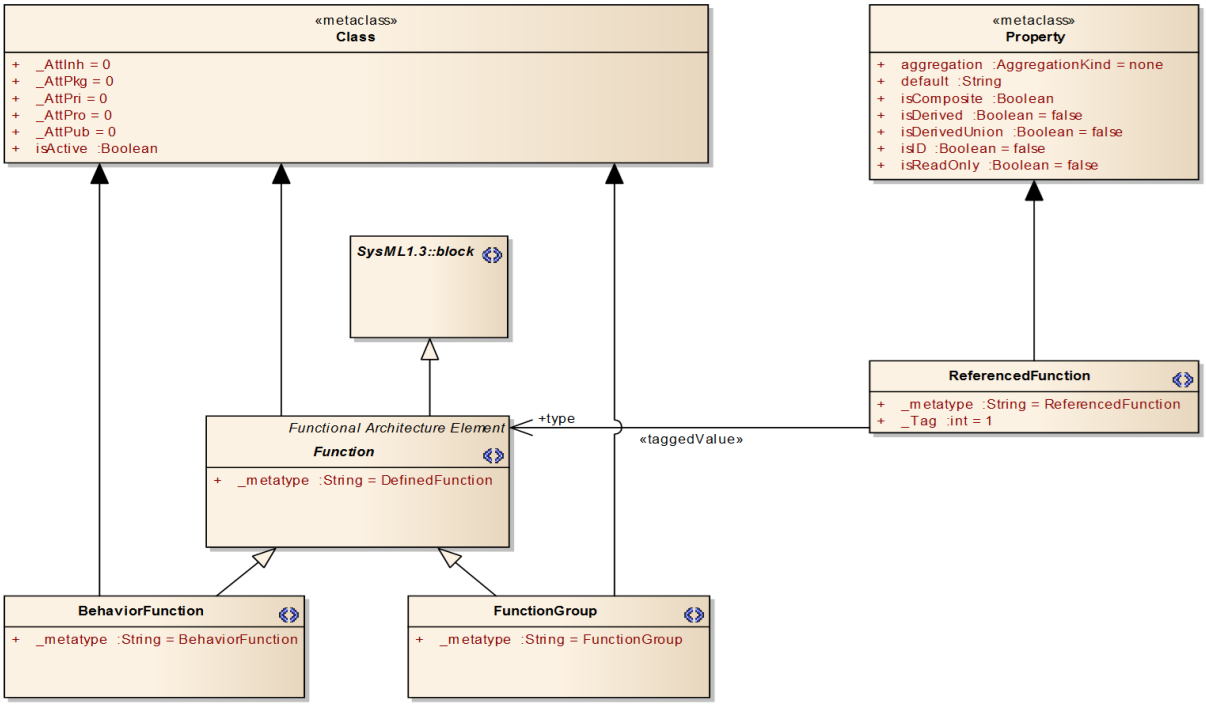

Figure 3. Metamodel elements to support the specification of Function definition.

An excerpt of the meta-model of the logical view is shown in Figure 4. It defines logical components of the system. A logical component realizes one or multiple system functions that are mapped via the *RealizedBy* relation to a logical component. Components are either behavior components that describe system behavior, or aggregated components that are defined by a substructure of components. This view also does not distinguish between hardware and software components, but enables already the definition of component interfaces with communication semantics.

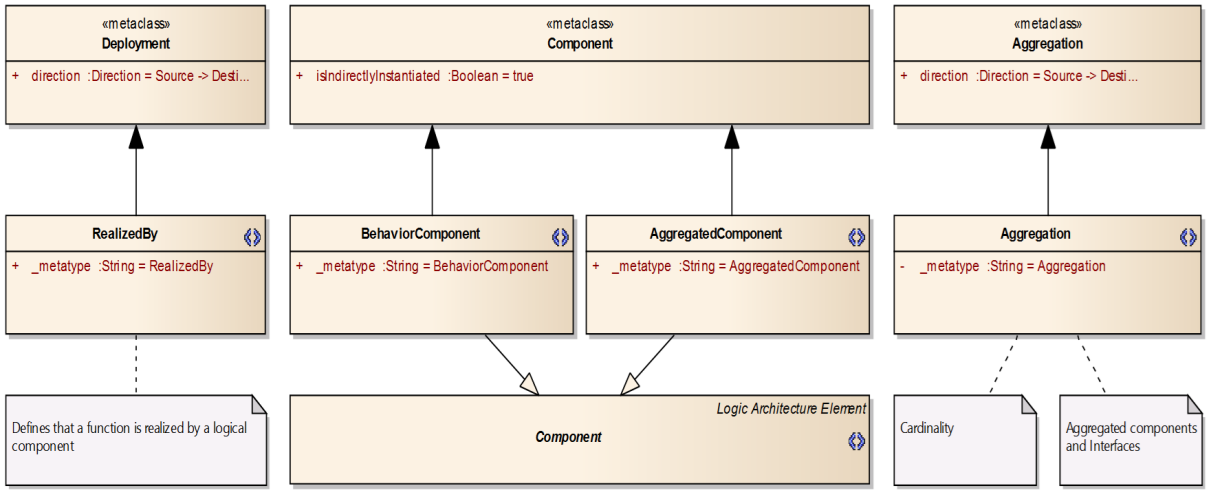

Figure 4. Metamodel elements to support the specification of logical components.

The central elements in the technical/deployment view are the concrete hardware and software entities (e.g. processors, networks, busses, source code files) and the mapping of logical components to these entities. It enables the definition of device and network types, as well as the specification of the application architecture on code level that is used for traceability purposes and for architecture compliance checking. Deployments are represented by tasks, which may be triggered periodically, or explicitly via events.

Early Detection of Design Faults

One of the goals of model-driven development is to enable early detection of design faults before implementing them with source code. This is why more and more the development of safety critical systems have been centered in models. For example, the use of Component Fault Trees as means to analyze how components failures contribute to a hazard occurrence, and model-based specifications of safety requirements. Two aspects that have been widely discussed are incompleteness and inconsistency of safety-critical systems specifications. This is particularly important because, if any of the traces that are supposed to exist among these elements are missing or inconsistent, there will be a lack of evidences to justify that all the potential failures of a safety-critical system were properly analyzed and mitigated.

By definition, completeness is ensured when all the information necessary for a problem definition or justification is found within the specification [9][8][4]. Consistency relates to any situation in which two or more portions of a specification obey relationships that should exist between them [9][8]. Precise completeness and consistency checks can only be performed when the different systems artifacts are properly modelled. For example, Antonino and Trapp [2] proposed the safety requirements decomposition pattern (cf. Figure 8), which is a decomposition structure to support the creation of model based safety requirements

specification that are traceable to architecture models and failure propagation models. In this regard, the embedded modelling profile presented in this paper plays a key role, once it offers means to properly model architectures, and, consequently, collaborate for improving the analyzability of completeness and consistency of model based specifications of safety critical systems.

One example of a completeness check that can be performed is, for instance, checking if every safety requirement describes failures mitigations referencing, at least, one safety-critical logical element of the architecture. One example of consistency check that can be done is with respect to safety integrity levels, more specifically, check if the safety requirements are addressed by safety-critical architecture elements with an equal or more stringent safety integrity level. We have implemented these checks as part of the too Fraunhofer I-SafE (Integrated Safety Engineering), and have collected evidences that the combined use of the Safety Requirements Decomposition Pattern and the embedded modelling profile have been a key aspect for improving completeness and consistency of model-based safety critical systems specifications.

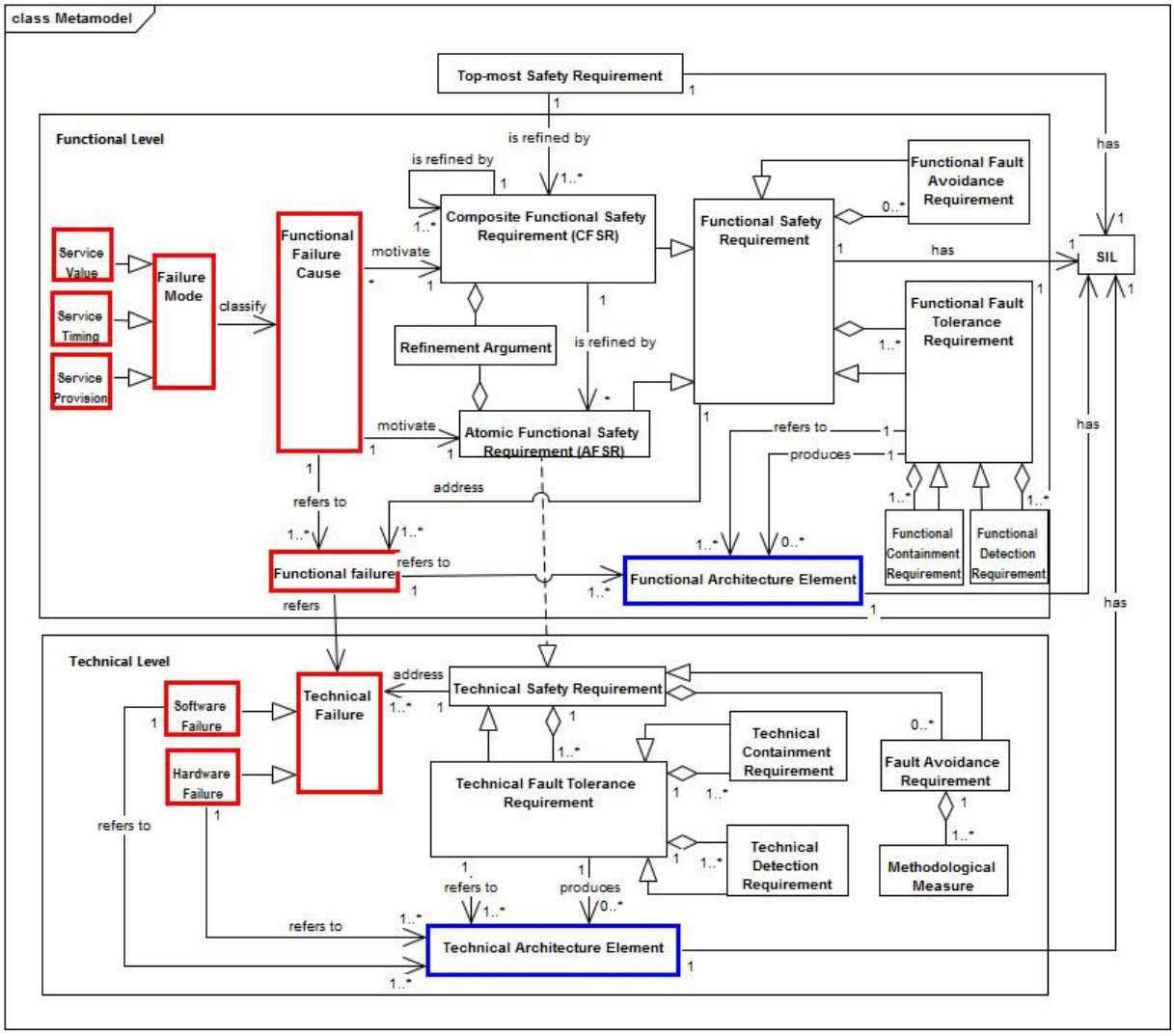

Figure 5. Safety Requirements Decomposition Pattern.

Conclusion

In this paper, we have presented an overview on our embedded systems modeling profile that extends the UML and SysML notations with explicit modeling constructs for embedded systems. It enables the definition of function networks, logical architectures, technical/Deployment architectures, as well as the existing SysML views including sequence diagrams and state charts. With tailored modeling and analysis tools, numerous design flaws like inconsistent models or non-traceable requirements can be located easily.

References

[1] K. Pohl, H. Hoenninger, R. Achatz and M. Broy, Model-Based Engineering of Embedded Systems - The SPES 2020 Methodology, Springer, 2012.

[2] P. O. Antonino and M. Trapp, "Improving Consistency Checks between Safety Concepts and View Based. Architecture Design," in PSAM12 - Probabilistic Safety Assessment and Management Conference, Honolulu, Hawaii, USA, 2014.

[3] N. Rozanski and E. Woods, Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional, 2005.

[4] N. G. Leveson, "Completeness in formal specification language design for processcontrol systems," in Third Workshop on Formal Methods in Software Practice, Portland, Oregon, 2000.

[5] L. Fuentes-Fernandez and A. Vallecillo-Moreno, “An introduction to uml profiles,” Journal of UML and Model Engineering, vol. 2, 2004.

[6] P. Liggesmeyer and M. Trapp, “Trends in embedded software engineering,” IEEE Software, vol. 26, no. 3, pp. 19–25, May 2009.

[7] J. Rumbaugh, I. Jacobson, and G. Booch, Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education, 2004.

[8] D. Zowghi and V. Gervasi, "On the Interplay between Consistency, Completeness, and Correctness in Requirements Evolution," Journal of Information and Software Technology, vol. 46, no. 11, pp. 763-779, 2004.

[9] D. Zowghi and V. Gervasi, "The Three Cs of Requirements: Consistency, Completeness, and Correctness," in 8th International Workshop on Requirements Engineering: Foundation for Software Quality, (REFSQ'02, Essen, Germany, 2002.

Authors

Dr. Thomas Kuhn received his PhD from Kaiserslautern University in 2009, where he was working as a researcher since 2004. Since 2008, he is working for Fraunhofer IESE. Currently, he is heading the Embedded Software Development department, responsible for architecture analysis approaches and virtual prototypes that enable early decision support based on measurable facts.

Pablo Oliveira Antonino works as computer scientist at the Fraunhofer IESE since 2009. His work focuses on the construction and analysis of safety-critical systems architectures, mainly in the Automotive, Agriculture and Avionics domains.

Kontaktinformationen:
Thomas Kuhn / Pablo Antonino
Fraunhofer IESE
Fraunhofer-Platz 1
67663 Kaiserslautern

thomas.kuhn@iese.fraunhofer.de
pablo.antonino@iese.fraunhofer.de