Automotive Software Architecture Views and Why we need a new one – Safety view

Miroslaw Staron

Automotive Software Architecture Views and Why we need a new one – Safety view

Miroslaw Staron*
*University of Gothenburg, Sweden
miroslaw.staron@gu.se

***Abstract*—The growth of the size and complexity of automotive software has been driven by the increased number of software-dependent innovations in modern cars. The growth of the size and complexity drives the usage of multiple abstraction levels in automotive software designs. One of these levels is the architecture level where the high level design of software is documented using multiple views such as logical, function, deployment.**

In this paper we review these architectural views and argument that they require one more view – safety view – which is compliant with the ISO/IEC 26262 and links the design of software systems (both the functional and system views) with the safety goals. We advocate the need for this view that would help the designers to understand and trace the design of software safety throughout the software lifecycle.

I. Introduction

As the amount of software in modern cars grows we observe the need to use more advanced software engineering methods and tools to handle the complexity, size and criticality of the software [1], [2]. We increase the level of automation and increase the speed of delivery of software components. We also constantly evolve software systems and their designs in order to be able to keep up with the speed of the changes in requirements in automotive software projects.

However, we also need to handle the complexity is an efficient way, meaning that we need to adjust the engineering methods depending on the criticality of the software, according to the prescriptions of the ISO/IEC 26262 standard [3] – both during the design and verification/validation of software systems. These analyses are currently poorly supported in the general software engineering tooling and methodologies. Therefore in this paper we set off to address the following research question:

How to provide explicit support for safety-related design elements in the architecture work?

By the safety-related design we mean the support for safety related questions which are important during the design of the system, such as:

- *How to identify which software elements should be validated using which methods based on their ASIL levels?*
- *How to find out which elements are linked to a specific safety case?*
- *How to identify which elements should be re-validated if the safety goals/cases change?*

Even though these questions seem rather straightforward we need an architectural view which will allow us to see the ASIL levels assigned to the system elements, their verification requirements, relevant safety-related metrics (e.g. control path complexity), etc. We also need to be able to see which software elements have none of the safety elements assigned to them, such as which software components have no ASIL level assigned or which ASIL D software components have a control path complexity way above verifiable limits.

II. Related work

One of the most prominent works in the area of visual safety argumentation has been done by Kelly and Weaver [4] who presented a notation for modelling safety goals. The main goal of the notation is to link the safety goals to the safety evidence using the safety argumentation. Similar approach has also been done in the later work of Hawkins et al. [5]. In our proposal we use the same symbols for the safety arguments and goals, and complete with the other modeling elements of interest.

Xiaoping et al. [6] describes the safety editor based on the ISO/IEC 26262 standard and shows that this view requires linking to the architectural elements of the system construction, something that we also support and present a proposal in this paper.

Hause and Thom [7] present a method for extending SysML to show safety concerns integrated into SysML models. Our proposal is based on very similar principles, although adding a separete view, which is not present in SysML.

Another part of the safety view is the visualization of metrics in order to reason about the sufficiency of the safety argumentation. Termeer et al. [8] presented a tool which augmented UML diagrams with metric information, a view which we intend to integrate with the safety view of the architecture – or at least the ideas presented in their work.

III. Architectural views

Architecting is a continuous process, which starts at the first requirement and ends with the last defect-fix in the car. Although one could see the process of architecting as a prescriptive design, the process continuous as the design evolves. Certain aspects of design decisions influence the archtiecture and are impossible to be known a priori – increased processing power required to fulfill late function requirements or safety criticality of the designed system. If not managed correctly the architecture has a tendency to evolve into a descriptive documentation that needs to be kept consistent with the software itself [9], [10], [11].

In order to put the process of architecting in a context and describe the current architectural views in automotive software archtiectures let us first discuss the V-model as shown in Figure 1

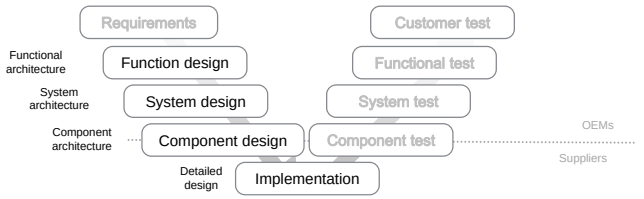

Fig. 1. V-model with the focus on architectural views and evolution.

The first level is the functional development level where we encounter two types of the architectural views – the functional view and the logical system view. The functional view, often abbreviated to the *functional architecture* is the view where the focus is on the functions of the vehicle and their dependency on one another [12]. An example of such a view is shown in Figure 2.

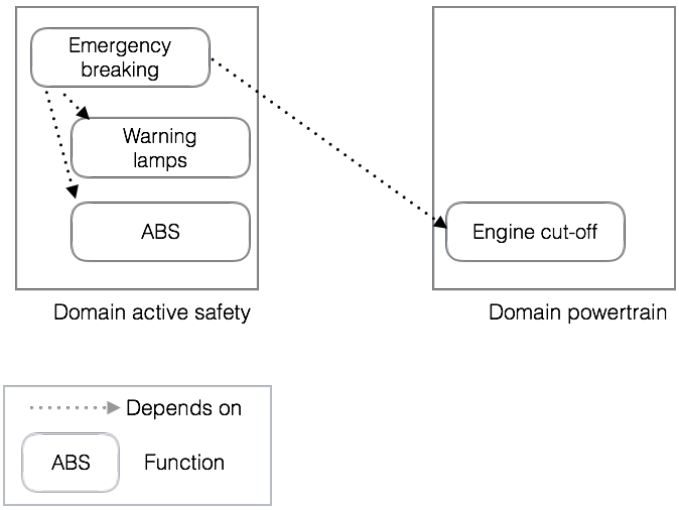

Fig. 2. Example of a functional architecture – or a functional view.

Another view is the system view on the archtiecture, usually portrayed as a view of the entire electrical system at the top level with accompanying lower level diagrams (e.g. class diagrams in UML). Such an overview level is presented in Figure 3.

The focus of the view is on the topology of the system. This view is often accompanied by the logical component architecture as presented in Figure 4.

For the logical view the architects often use a variety of diagrams (e.g. communication diagrams, class diagrams, component diagrams) to show various levels of abstraction of the software of the car – usually in its context. For the detailed design these architectural models are complemented with low level executable models such as Matlab/Simulink defining the behaviour of the software [13].

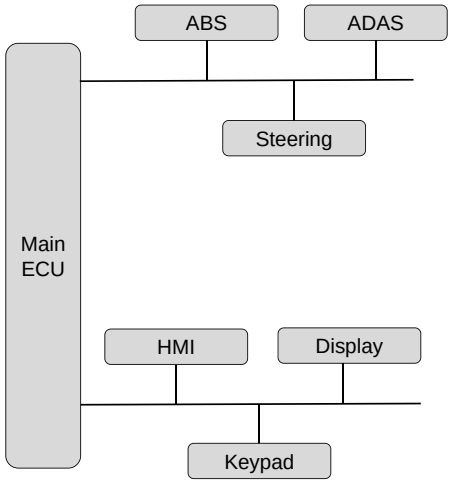

Fig. 3. Example of a system architecture – or a system view.

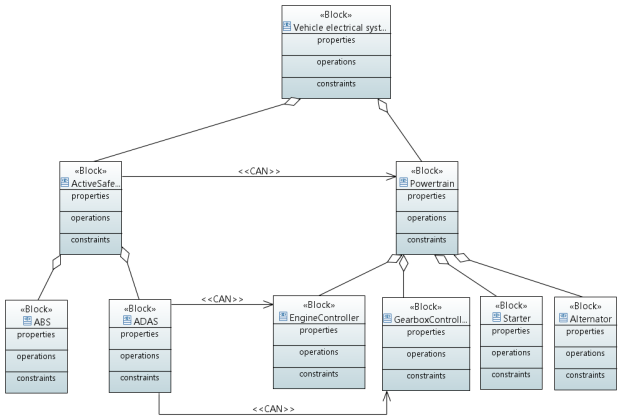

Fig. 4. Example of a logical view – a UML class diagram notation.

IV. Safety view

The previously presented view are dedicated for the design disciplines in software engineering, but in the automotive software engineering the safety aspects are usually treated as a separate concern and therefore we postulate that they need a specific architectural view. Having such a view is important as it allows to easily trace the safety argumentation, goals and elements to the system design elements. And since the number of system elements in modern cars is constantly growing, having such a linkage allows to assure that we do not miss an important design step or validation step/method required for achieving a specific safety level.

In this view we need to include the concepts which are important for safety engineering:

- Safety goals – safety goals as specified in ISO/IEC 26262
- Safety arguments – argumentation for the fulfillment of the safety cases (documents), usually in form of the safety cases
- Safety verification requirements – verification require-

ments as prescribed by the ASIL levels in ISO/IEC 26262

- Safety verification methods – verification methods applied to the system elements
- System elements – elements of the system (taken directly from the other views) which are linked to safety goals, arguments and verification requirements/methods.

A notation for modelling only the safety part has already been proposed and designed by Kelly et al. [4] and we postulate that it needs to be combined with the software engineering notations in order to provide a more holistic view on the development of safety critical software systems.

A. Modelling of the view

Let us first present the meta-model of the notation, i.e. the abstract syntax of the modelling language. Once we presented the meta-model we present the concrete syntax and then we present an example of using this notation.

The abstract syntax (metamodel) is presented in Figure 5 where we use three colors to designate elements from three domains. For the safety notation of Kelly et al. [4] we use the white color, for the component design (UML block diagram notation) we use the orange color and for the added elements of the safety view we use the blue color. For the sake of simplicity of the argumentation we do not focus on the entire notation of Kelly et al. nor the UML notation. By this meta-model we intend to show that the linkage of between these two diagrams can be straightforward and the only additions are the two elements – the abstract SafetyViewElement which is the top hierarchy class for all elements in the new safety view; and the SafetyAssociation which associates the safety elements with the component design elements.

As we can observe in the meta-model, the missing association is bi-directional (both source and target) which allows to search for the elements from one another. This ability is very important when producing reports regarding safety elements and for the traceability between safety argumentation and design components.

We decided to place the association on the relatively high level of the hierarchy as it allows for flexibility on linking different elements from these two domains. We perceive this flexibility as an important aspects of the design. We also need more experimental results from modelling of safety cases in order to validate this design decision.

The example of the notation is presented in Figure 6 where we show how to add the link (in form of a dotted line) between the safety solution and the construction element. The goal presented in this example is very simple, but illustrates the principle.

V. Conclusions

In this paper we presented a position where we advocate adding one additional architectural view – *Safety view* – which combines the elements of safety argumentation and the elements of the construction (design) of the software systems which are affected by these safety elements.

The research results presented in this paper contain the meta-model of the new view which shows how the combination of these two domains (safety and software construction/design) can be combined by a simplistic extension of the UML/SySML meta-model. This extension can be done both as a fully-fledged extension or as a stereotype extension [14], [15]. The stage of this research is rather preliminary but it shows the potential impact on the design of software for software critical systems.

Our future directions include the development of the UML profiles for Papyrus UML tool where we can include the stereotypes for modelling safety cases and linking them to the software design elements as proposed in this paper.

Acknowledgment

The research presented here is done under the VISEE project which is funded by Vinnova and Volvo Cars jointly under the FFI program (VISEE, Project No: 2011-04438)

References

[1] M. Staron, "Software complexity metrics in general and in the context of iso 26262 software verification requirements." in *Scandinavian Conference on Systems Safety*. http://gup.ub.gu.se/records/fulltext/233026/233026.pdf, 2016.

[2] S. Fürst, "Challenges in the design of automotive software," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 256–258.

[3] I. ISO, "26262–road vehicles-functional safety," *International Standard ISO/FDIS*, vol. 26262, 2011.

[4] T. Kelly and R. Weaver, "The goal structuring notation–a safety argument notation," in *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*. Citeseer, 2004.

[5] R. Hawkins, T. Kelly, J. Knight, and P. Graydon, "A new approach to creating clear safety arguments," in *Advances in Systems Safety*. Springer, 2011, pp. 3–23.

[6] Y. Luo, Z. Li, and M. Van Den Brand, "Development of a safety case editor with assessment features," in *Proceedings of the 2nd Workshop on Automotive Software/Systems Engineering*, pp. 1–4.

[7] M. C. Hause and F. Thom, "An integrated safety strategy to model driven development with sysml," in *System Safety, 2007 2nd Institution of Engineering and Technology International Conference on*. IET, 2007, pp. 124–129.

[8] M. Termeer, C. F. Lange, A. Telea, and M. R. Chaudron, "Visual exploration of combined architectural and metric information," in *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on*. IEEE, 2005, pp. 1–6.

[9] U. Eliasson, R. Heldal, P. Pelliccione, and J. Lantz, "Architecting in the automotive domain: Descriptive vs prescriptive architecture," in *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*. IEEE, 2015, pp. 115–118.

[10] A. Shahrokni, P. Gergely, J. Söderberg, and P. Pelliccione, "Organic evolution of development organizations-an experience report," SAE Technical Paper, Tech. Rep., 2016.

[11] L. Kuzniarz and M. Staron, "Inconsistencies in student designs," in *the Proceedings of The 2nd Workshop on Consistency Problems in UML-based Software Development, San Francisco, CA*, 2003, pp. 9–18.

[12] A. Vogelsanag and S. Fuhrmann, "Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study," in *Requirements Engineering Conference (RE), 2013 21st IEEE International*. IEEE, 2013, pp. 267–272.

[13] J. Friedman, "Matlab/simulink for automotive systems design," in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*. European Design and Automation Association, 2006, pp. 87–88.

[14] M. Staron and C. Wohlin, "An industrial case study on the choice between language customization mechanisms," in *Product-Focused Software Process Improvement*. Springer, 2006, pp. 177–191.

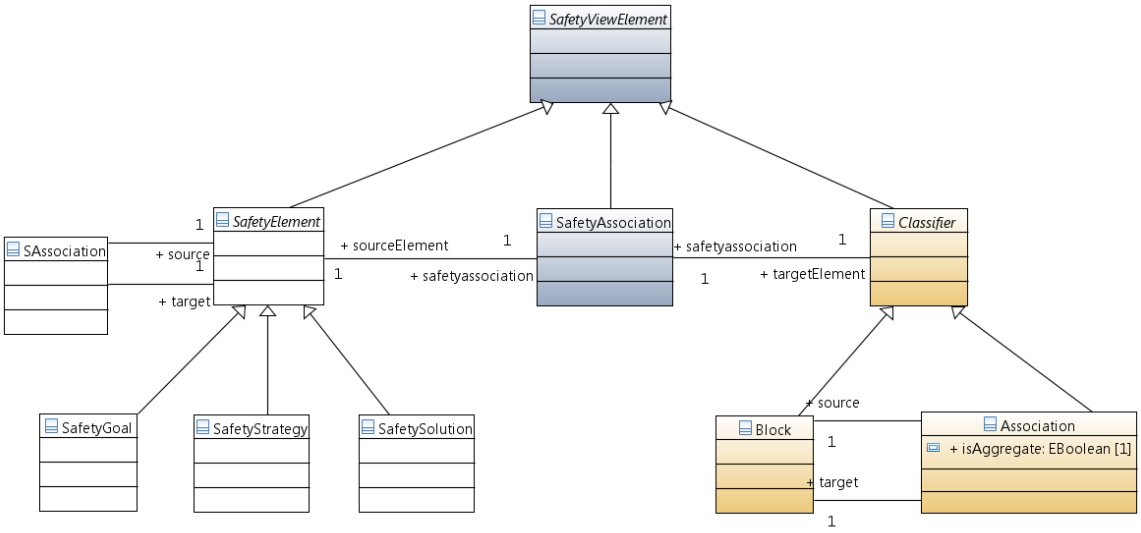

Fig. 5. The meta-model of the safety view.

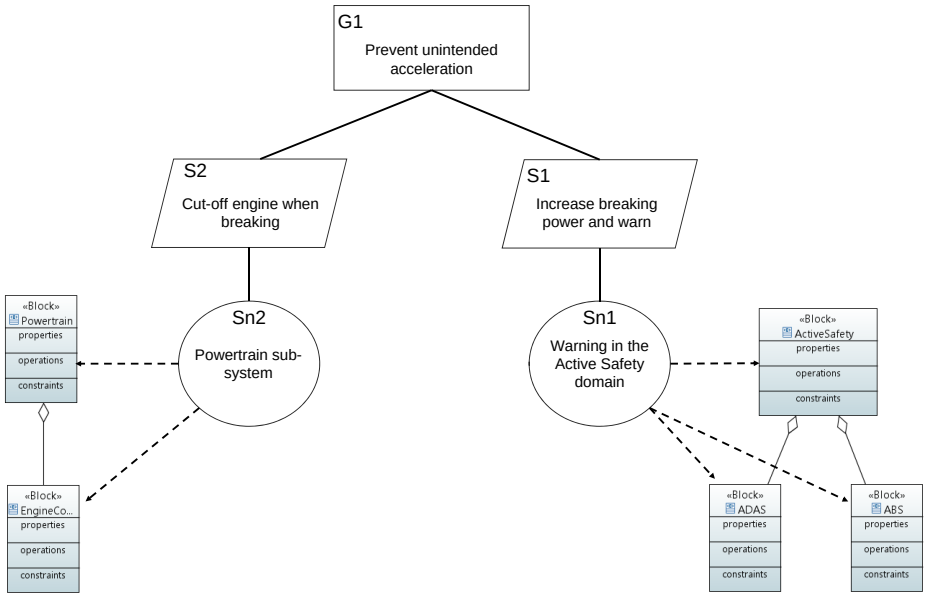

Fig. 6. Example of a safety view combining two notations.

[15] M. Staron and L. Kuzniarz, "Properties of stereotypes from the perspective of their role in designs," in *Model Driven Engineering Languages and Systems*. Springer, 2005, pp. 201–216.