embeddedworld2021 Exhibition&Conference ...it's a smarter world DIGITAL

Designing Safety In by Extending System Modeling Languages

Juha-Pekka Tolvanen
MetaCase
Jyväskylä, Finland
jpt@metacase.com

Steven Kelly
MetaCase
Jyväskylä, Finland
stevek@metacase.com

***Abstract*—To create safe systems, systems development should follow applicable safety standards. Nowadays much systems development uses modeling languages, yet most of these languages do not recognize safety concerns adequately. We present an approach in which relevant safety aspects are treated as first-class citizens: The modeling language includes concepts specifically for safety, in addition to those for other aspects of system design. In this paper we present how an automotive safety standard, namely ISO 26262 for functional safety, has been added to a system modeling language, extending its metamodel and the related tool support to enable natural modeling of safety aspects, collaborative development and safety reporting along failure analysis. The extended safety modeling support shows the key benefits of addressing safety together with system design: traceability between system design and safety design enables collaboration between system engineers and safety engineers, and error-prone, time-consuming manual phases are replaced with automated design transformation and analysis.**

Keywords—safety; domain-specific language; ISO 26262; Model Based Development

I. INTRODUCTION

To create safe systems, systems development should follow applicable safety standards such as ISO 26262 [4] or SOTIF/ ISO 21448 [5]. Nowadays much systems development uses modeling languages, yet most of these languages do not recognize safety concerns adequately or guide towards design of safe systems (e.g. SysML [10]). We present an approach in which relevant safety aspects are treated as first-class citizens: The modeling language includes concepts specifically for safety – and linked with those dealing with other aspects of system design. Modeling of safety systems is thus not only about system blocks, signals and client/server connections, but also about hazards, hazardous events and safety goals. This makes safety aspects concrete and visible to the whole team. As the safety aspects are linked with the rest of the system development language, the approach also enables collaboration among system engineers and safety engineers as well as provides the basis for the necessary traceability. The traceability ranges from the logical dependencies among requirements, to the satisfaction mappings from requirements to system designs, and to the evaluation mappings from requirements to verification and validation cases. Capturing safety information in models lets us use it in tool support and automation, e.g. creation of Fault Tree Analysis (FTA), Failure Models and Effects Analyses (FMEA), and safety-related documentation ([1][8]).

In this paper we present how safety aspects can be added to a system modeling language, looking at both the concepts and the process of adding them. To demonstrate the approach, we take a concrete example from automotive safety, namely the ISO 26262 functional safety standard [4], and add it to a system modeling language. The same process can be applied for other safety standards: we have also applied it to ISO 13849-1, safety for machinery.

The process starts by extending a metamodel of a modeling language with safety concepts, along with their rules for things like naming conventions and connections. This is followed by adding visual notations to represent the new safety modeling concepts. Having a notation allows the language to be tested with reference examples and demonstrated to language users. Automated tool support is also added for verifying model correctness and offering traceability of safety design with the rest of system design. Generators are made to read the models and provide a link to existing analysis and simulation tools, enabling automatic creation of FTA, FMEA, and safety-related documentation.

Most importantly, extending the language is an incremental process, in which the evolving modeling support can be tested and validated with language users by using it to specify actual systems. In other words, the process from adding a language concept in the metamodel to its use is seamless – it takes just a few minutes to add a safety concept to the metamodel along with its notation and rules, and it is instantly available to users for them to provide feedback for further language definition. We conclude the paper by showing the benefits of the safety modeling support with practical examples: traceability between system design and safety design enables collaboration between system engineers and safety engineers, and error-prone, time-consuming manual phases are replaced with automated design transformation and analysis.

II. Language Design

Today, companies in many fields are required to follow safety standards, such as ISO 26262 on functional safety in the automotive industry. These standards typically define a relevant vocabulary and concepts along with rules and processes for safety design. Using ISO 26262 as an example, some of the key concepts are:

- A Safety Goal is a top-level safety requirement that is assigned to a system, with the purpose of reducing the risk of one or more hazardous events to a tolerable level.
- A Hazardous Event is a relevant combination of a vehicle-level hazard and an operational situation of the vehicle with potential to lead to an accident.
- A Hazard represents a condition or state in the system that may contribute to accidents.
- A Safety concept defines means to achieve the safety goal.
- An Item is a particular system product that identifies the scope of system information and its top-level system functional requirements.

Along with these concepts there are also rules and process guidelines to ensure that all aspects of safety are considered. For example, for each Safety Goal one or more safe states should be defined: the Safety Goal must address one or more Hazardous Events and it must be linked to one or more Requirements. System engineers and safety engineers need to master these concepts and all their related constraints to perform safety design and document it accordingly (for acceptance). Unfortunately, having a generic and universal set of safety concepts is not possible: safety standards are not compatible, have different meanings and even follow opposite values (e.g. as for safety integrity level defining risk-reduction levels in different safety standards). Many safety standards are tailored to be specific to the industry, with different standards for machinery, railway, industrial processes, automotive etc.

A. *Abstract syntax*

Adding safety to a modeling language starts by defining the relevant safety concepts as a metamodel: Fig. 1 shows the relevant concepts of ISO 26262 defined for Dependability modeling. Top left is the definition of Safety Goal along with its properties, such as its Safety level and its SafeState (a Mode object), and connections to other concepts like Requirements and the Hazardous Events it addresses.

We need to integrate these concepts with our existing system modeling language. In our case, we extended EAST-ADL, a system modeling language supporting aspects relevant for automotive systems, like product lines with variants and automotive software architecture AUTOSAR [1]. The Requirement concept shown in the ISO 26262 metamodel is also an existing modeling concept of EAST-ADL and we reuse it here. The Item concept in ISO 26262 has a property 'Vehicle features' referring to a collection of Features, and EAST-ADL also defines a concept of Feature with similar semantics, so we link to that.

In our language extension work we used MetaEdit+ [6], as it already provides metamodels of EAST-ADL (and UML, BPMN etc.) that can be directly modified and extended, without breaking existing models. In addition to Dependability as in Fig. 1, we also defined a metamodel for error modeling that allows to specify possible incorrect behaviors of a system in its operation (e.g. component errors and their propagations). An example of error modeling is shown in Fig. 4.

B. *Concrete syntax*

Each modeling concept, such as elements, their connections or individual properties, has a visual representation so that humans can create, read and validate the models. For this

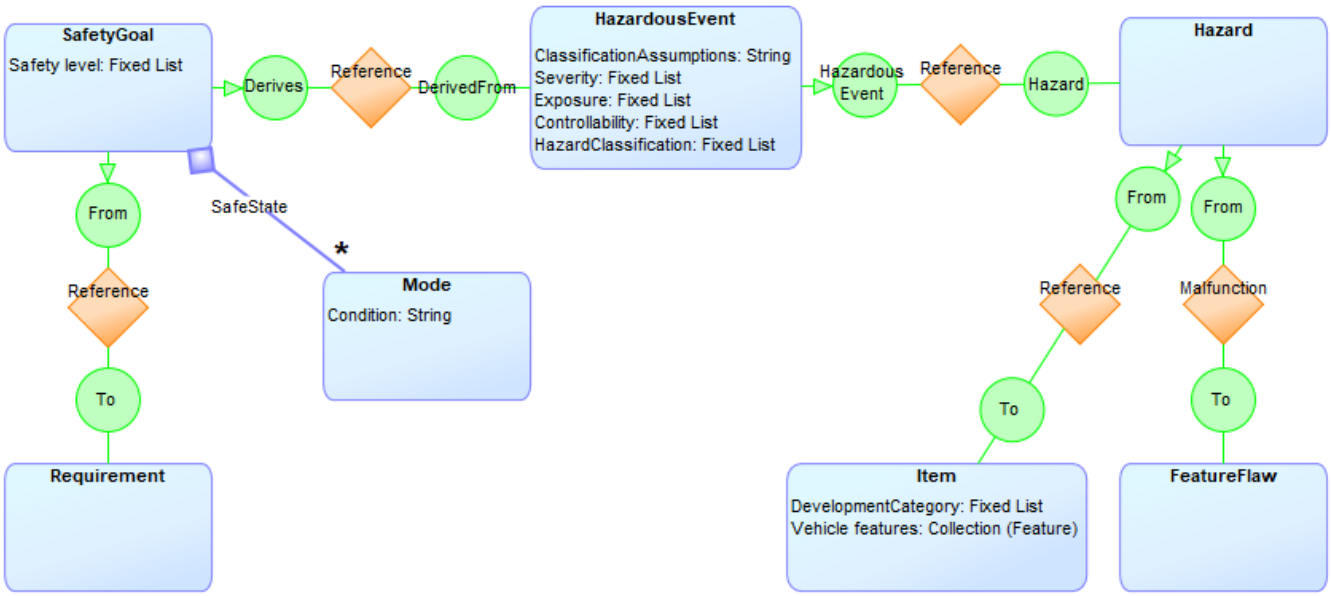

Fig. 1. Safety concepts of ISO 26262 as metamodel (partial)

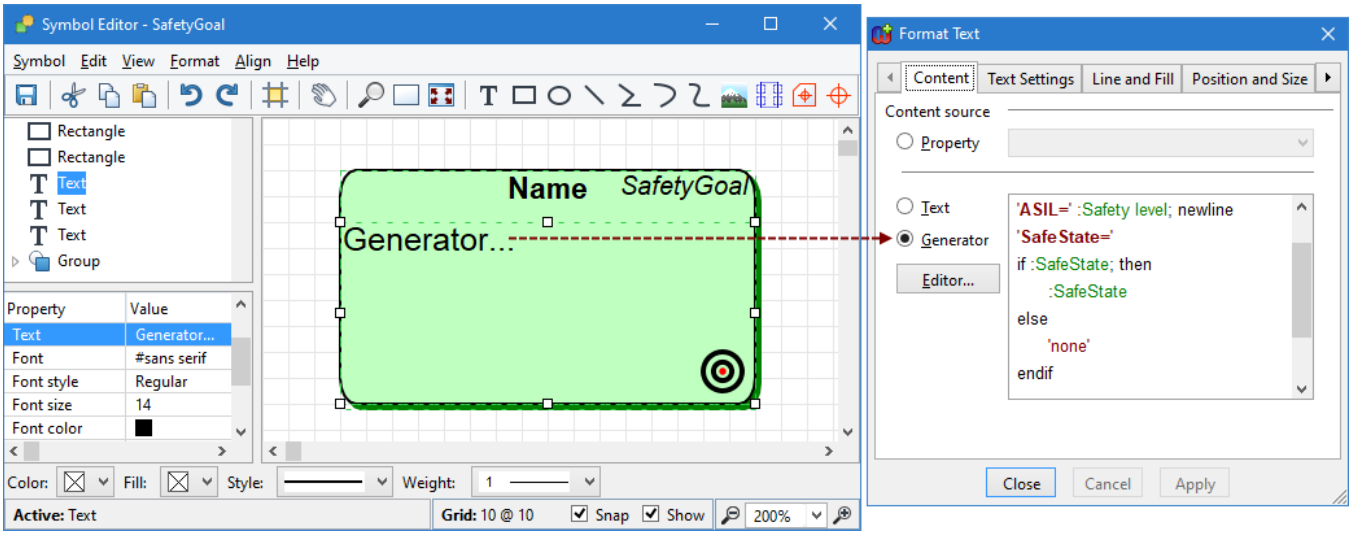

Fig. 2. Notation for Safety Goal

purpose, we add notational symbols for them. Fig. 2 illustrates a notation for Safety Goal defined in the Symbol Editor of MetaEdit+. The symbol may also have conditional or dynamic parts for showing values, calculating them, or providing guidance for language user (e.g. that no Safe State is yet defined for a Safety Goal). Using different visualizations for different elements of safety (see e.g. Fig. 3 and Fig. 4) makes models more readable – as opposed to having different kinds of things all represented by the same kind of rectangular block [7].

C. *Rules and constraints*

The metamodel may also include constraints and rules that guide safety engineers. For example, to assist completeness the language checks if a Safety Goal is not related to any Requirements or is not derived from any Hazardous Event. Or, as a consistency check, Hazardous Events must have unique names and their ASIL classification only allows legal values (i.e. those in the standard).

D. *Generators*

A key part of MBSE's contribution to Systems Engineering is the use of automation to increase productivity and reduce time-consuming and error-prone tasks. We defined several generators to assist engineering, including:

- Create initial safety models directly from system models. Since both hardware specification and functional specification can then be utilized, safety engineers can get started more easily and ensure that safety designs are related to the planned system. (Compare this to defining the architecture again separately for the purpose of safety design – and keeping it in sync.)
- Traceability reports to assist collaboration among system engineers and safety engineers. MetaEdit+ allows collaborative modeling and versioning of system models and safety models, and the traceability reports provide dedicated views for engineers. Such traceability data is also needed for reporting safety work.
- Produce data for analysis tools, like performing FTA and FMEA analysis. This makes analysis easier with fast feedback loops when the system design is changed.
- Safety design reporting – as required by certification or company specific needs.

E. *Language definition process*

The language was implemented incrementally together with automotive engineers in language definition projects. Extensions to it have been made by companies internally (e.g. [8]) or together with external consultants. Experiences from industrial use of MetaEdit+ have shown that it takes on average two weeks to create specific modeling support for company specific needs (for various cases, see [9]).

III. Results

The created modeling support has been applied by OEMs and suppliers for various kind of systems such as motor control and ADAS systems. We illustrate its usage via safety design of Intelligent Speed Adaptation (ISA), a driving assistance system designed to help the driver to better comply with speed limits (for a detailed description of this system, see [2]). Fig. 3 illustrates the results of hazard analysis related to ISA providing a wrong speed limit: For this case two Hazardous Events are defined: one related to driving on a dual carriageway and another on a rural road. For both events Severity, Exposure, Controllability and ASIL values are defined as in ISO 26262. The dependability model also defined the safety goal that reduces the ASIL level to an acceptable level (QM) with Requirement #14 stating that speed limit changes must be acknowledged by the driver. The concrete syntax of safety goal defined in Fig. 2 is illustrated in use at modeling time in bottom left of Fig. 3.

These dependability models are integrated with the rest of the designs, as here with Requirement #14 and Item to Vehicle feature called Intelligent Speed Adaptation. This feature is part of the vehicle feature model specifying all possible features that a vehicle in a product line may have. Being linked to features thus allows to trace implications of safety related features to all

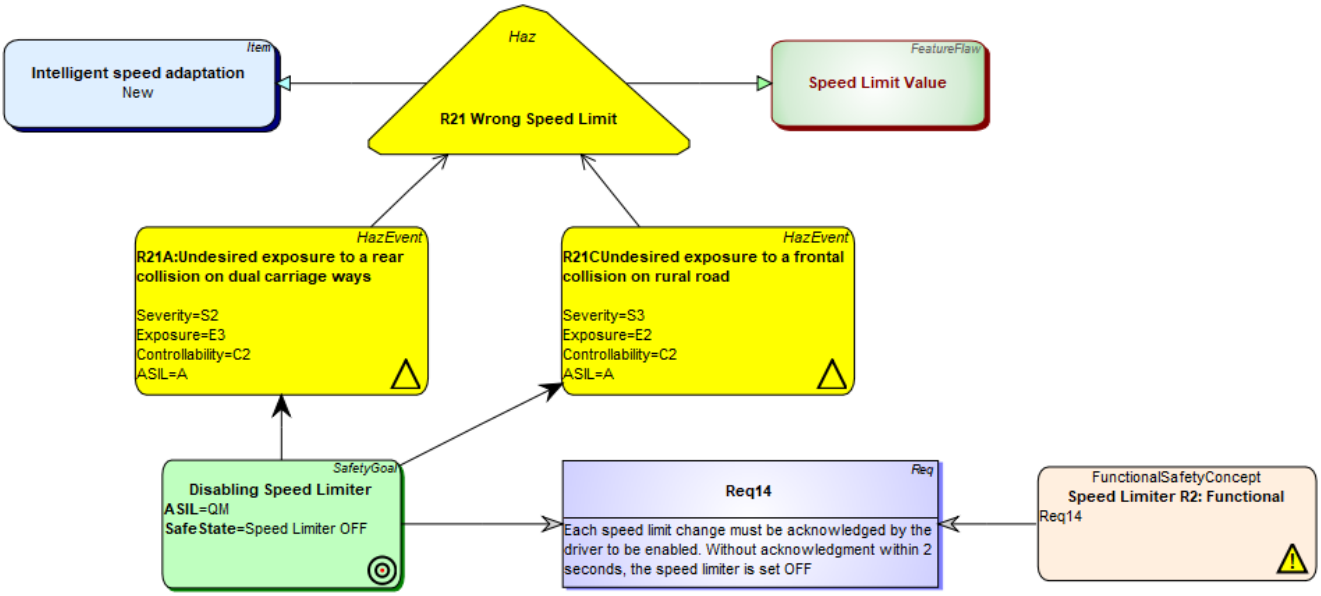

Fig 3. Dependability model specifying safety

possible vehicles they are applied in, rather than just one particular vehicle model and its configuration.

The same applies also for requirements: Engineers can trace from all Requirements to see if they are used to meet Safety Goals, and produce documentation such as reporting Safety Goals together with related Requirements (as done in [8]). They can also collaborate and version the specifications together [6]. For example, suggested changes based on safety can be traced back automatically to related system design to be changed based

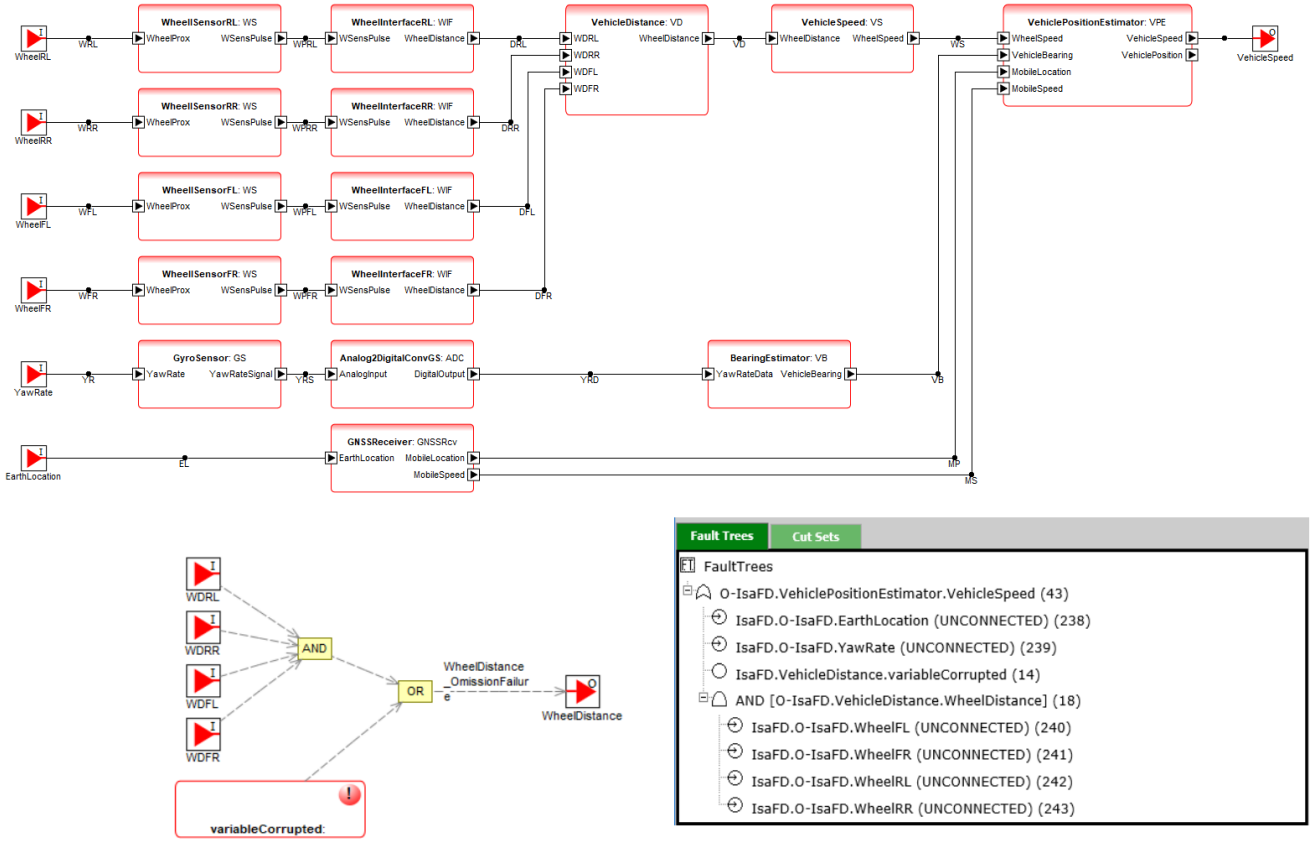

Fig. 4. Error model for ISA and its VehicleDistance component along with related FTA

on safety analysis. This facilitates feedback and change iterations during development of safe systems.

System designs are also applied as a basis for error modeling. The left part of Fig. 4 shows the error model of the ISA system and the error logic of one of its components: omission of Vehicle distance from the system component can occur if it does not get data from the wheel sensors or the component has an internal error. From such error models automated FTA and FMEA can be performed as shown on the right side of Fig. 4: a fragment of a fault tree as presented by the HiP-HOPS tool [3]. For any system of a realistic size, the automated analysis quickly becomes an invaluable tool. Error models can be similarly translated to other FTA/FMEA tools and, depending on the tools used, the analysis can be further extended by providing information on failure rates and repair rates.

IV. Conclusions

Dedicated and precise support for safety standards can be achieved by extending a system modeling language to support them. This language-driven approach enables collaboration between system engineering and safety engineering, improving productivity and quality as safety work – dependability and error models – can be linked with the wider system. This also removes error-prone tasks and steps such as manually creating safety models, keeping them in sync with the system design, or performing FTA and FMEA analysis manually.

We have also created modeling support for other standards (e.g. ISO 13849-1) and analysis tools. The same principles of language extension (metamodel, constraints, notation, generators) were applied similarly with just the implementation varying: e.g. generators target different analysis tools and their formats, or models include different safety concepts (e.g. 'SafetyMeasure') and provide support for company specific reporting. Creating such modeling support with editors and links to analysis tools usually takes a few man-weeks with MetaEdit+.

References

[1] H. Blom, D. Chen, K. Kaijser, H. Lönn, Y. Papadopoulos, M. Reiser, R.T. Kolagari, S. Tucci, "EAST-ADL: An Architecture Description Language for Automotive Software-intensive Systems in the Light of Recent use and Research". In: International Journal of System Dynamics Applications, 2016

[2] J. Ehrlich, J.-P. Tolvanen, "Modelling with EAST-ADL: Intelligent Speed Adaptation (ISA) as case study", FISITA 2016 World Automotive Congress, Busan, Korea, Sept. 26-30, 2016.

[3] HiP-HOPS [Online]. Available at: http://hip-hops.co.uk/ [Accessed Sept 2020].

[4] ISO Functional Safety, 26262-1, 2018

[5] ISO, ISO/PAS 21448, Road vehicles — Safety of the intended functionality, 2019

[6] MetaCase, MetaEdit+ User's Guide. [Online]. Available at: https://metacase.com/support/55/manuals/, 2018 [Accessed 4 Dec 2020].

[7] D. Moody, "The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," in IEEE Transactions on Software Engineering, vol. 35, no. 6, 2009.

[8] B. Sari, Fail-Operational Safety Architecture for ADAS/AD Systems and a Model-driven Approach for Dependent Failure Analysis. Springer, 2020.

[9] J.-P. Tolvanen, S. Kelly, "Effort Used to Create Domain-Specific Modeling Languages". In: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. ACM, 2018.

[10] Omg.org, System Modeling Language, version 1.6. [online] Available at: https://www.omg.org/spec/SysML/1.6/PDF, 2019 [Accessed 4 Sept 2020]