OpenCar: A SysML v2 Modeling Framework for Early Analysis of BoardNet Architectures

Sebastian Post, Johannes Koch, Aida Bevrnja, Christoph Grimm
University of Kaiserslautern-Landau, Chair of Cyber-Physical Systems
sebastian.post@rptu.de | johannes.koch@rptu.de | a.bevrnja@edu.rptu.de | cgrimm@rptu.de

***Abstract*—The paper describes a modeling framework for automotive boardnet architectures: OpenCar. The modeling framework is based on the SysML v2 modeling language. By using a tool for constraint propagation, we enable the early estimation of key performances including latencies, throughput limitations, length, cost, and weight of the cable tree. The library in particular also provides means for safety and reliability analysis in line with ISO26262.**

***Index Terms*—Automotive boardnet, architecture, SysML v2**

I. Introduction

Today's vehicles have an increasing amount of software and AI capabilities for multimedia functions and autonomous driving. In particular autonomous driving needs data from sensors located at various points in the vehicle and multiple computing units (CU) that can be located throughout the entire vehicle. These sensors and CUs communicate via the boardnet.

In the past, there was a 1:1 mapping between the functions and CUs within the vehicle. By grouping similar features from a specific domain to a controller, e.g. for the power train, radar domain, etc. one gets a *domain architecture*. However, the increasing complexity of domain architectures leads to the increased length, cost, and weight of the boardnet wires: the cable tree can be as long as 5 km and weigh up to 80 kg, which makes it one of the heaviest components in a vehicle[1].

To address this problem, different functions, i.e., radar, and ultrasonic in the same topological zone of the car use a common CU in the zone (i.e., front zone). Such *zonal architectures* potentially permit to reduce the cable length, weight, and cost. However, this requires optimization:

- Functional safety must be analyzed carefully because the CU and the used components must satisfy the strongest requirements of the functions using it — in the worst case, a CU hence would require Automotive Safety Integrity Level D (ASIL D) [2] safety.
- The development process changes. In simple 1:1 and domain architectures, a supplier was responsible for functions and the related CU. The mix of allocated functions in zonal or central architectures requires different collaboration between suppliers and OEMs considering functional safety and budgeting of data rates and latencies.

On the other hand, looking at the boardnet in an early stage allows optimizations that often lead to *hybrid architectures*. Such optimizations include:

This work was funded by the German BMBF within the projects GENIAL! under ref. no. 16ES0873 and KI4BoardNet under ref. no. 16ME0782.

- Taking advantage of redundant components (e.g., sensors) throughout the vehicle might reduce some components' ASIL level according to ISO 26262 [3].
- Sensor data fusion can in addition increase accuracy.
- Different architectures of the boardnet like rings, star networks, etc. might provide redundancy.

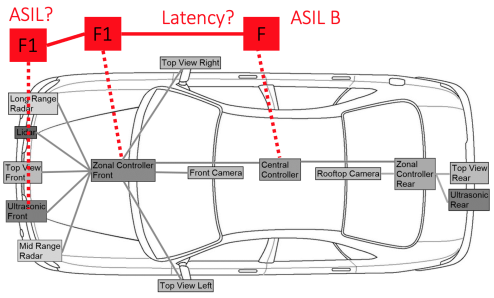

Fig. 1. Zonal Vehicle Architecture with Feature Mapping.

During the vehicle development, the OEM defines the features and manages the overall boardnet development. Suppliers suggest how to implement these features using hardware and software. To connect both processes, early and quick analysis of safety, latencies, weight, building spaces, power, heating, and costs as early as possible is needed Currently, this is often done using simple Excel sheets.

We introduce the OpenCar modeling framework that builds on top of the SysMD Notebook [4], [5]. The modeling framework OpenCar supports the model-based, collaborative optimization of boardnet architecture/topology and feature mapping at the early stages of the development of a boardnet architecture and topology by providing early and simple estimations of

- cable mass and cost,
- latencies and data rates along specified paths,
- safety and reliability analysis.

The novelty of the modeling framework is that it entirely builds on top of SysMLv2; hence, calculations and methods are independent from the underlying algorithms in the tool SysMD Notebook (constraint propagation). This makes the overall framework scalable and adaptive to more complex use cases. The reason is that often-needed extensions or adaptions can be handled at the level of models and do not require changing some algorithms for analysis inside a specific tool.

Furthermore, SysMLv2 models might be (re-)used within further development processes.

II. State of the Art and Related Work

Multiple attempts are made to model collaboration along the value chain. These efforts can be categorized into three main research areas. The first area focuses on enhancing collaboration within the value chain. The second area involves specific efforts to achieve this through standardization or tool interoperability. The third area involves analyzing and/or optimizing existing boardnet architectures.

The analysis of cooperation within the value chain is also considered by certain authors, e.g.,[6], [7], [8]. This cooperation is strongly requested by Soltani et al. They emphasize that such a strategy is currently missing and that such cooperation would be beneficial for the whole value chain. But this cooperation is a challenging scenario [9]: *"The need for establishing short feedback cycles [..]: While developing new functionality, basic software, and hardware, one should plan on how to receive feedback, which data to collect, and how to use it in the development."*

Some tools are also available targeting cooperation in the value chain. An example of such a tool is AutoSAR [8] which connects the OEM to the suppliers using this software. This has the advantage that requirements can be exchanged more easily. The definition of the *.kbl* file format [10] (currently .vec [7] is developed, which adds some features) allows a seamless exchange between OEM and suppliers, but these formats focus only on the physical description of the cable tree and not on the analysis of requirements. SysML v2 [11], [12] could be used to analyze the whole system because it has a standardized REST-API, which makes it easy to use by different stakeholders and improves collaboration in the value chain. However, to our knowledge, there are no tools and approaches in combination with SysML v2.

Many approaches show how to model safety and security aspects using a modeling language. Martin et al. [13] have developed a model-based systems engineering (MBSE) approach to model safety and security patterns using SysML v1. AltaRica [14] or PREEvision [15] are also used in some approaches to model safety and security. They all conclude that modeling safety and security aspects in a modeling language could improve the accuracy and efficiency of safety analysis in the automotive industry. Another framework focusing on properties such as weight latency and costs is described by Zheng et al. [16]. AADL is used for modeling the architecture, and the latencies are optimized for the task allocation. However, those approaches focus only on safety and security aspects but do not look at the whole vehicle to combine this analysis with other aspects like latency or weight analysis. This has the consequence that variants, uncertain values, and open design decisions can not be considered.

In [4] and [5] we have described an approach using SysML v2 for evaluating different architectures using the modeling language SysML v2 textual. We have created a tool called SysMD, which has a Jupyter Notebook[17] like GUI. This has the advantage of easy documentation of models and can be understood easier than models in graphical notation by beginners. The tool also supports bidirectional constraint propagation, so constraints are propagated throughout the model.

This work extends *CoDesign* with an OpenCar framework to model collaboration in the value chain. Furthermore, we show some applications of the framework.

III. Open Car modeling framework overview

The OpenCar modeling framework is based on the Automotive Architecture Framework (AAF) introduced by Broy et al. [18]. In natural language, the main relationships (italic) between elements (bold) are as follows: The modeling process starts with **requirements**, which describe the specific conditions the system must satisfy. These **requirements** are *satisfied by* **features** and **functions** which *are implemented by* **elements**. *elements* are

- **Software** that is *executed by* a **CU or ECU**, or
- **Sensors** that have the **function** to *sense quantities*, or
- **Actuators** that have the **function** to *perform actions*, or
- **Cables** that connect **components**

We use SysML v2 artifacts to model the above elements and relationships. *Part definitions* define a class, e.g., a class 'Car'. *Part (usages)* create instances of a class, e.g. 'myCar' of the class 'Car'. SysML v2 furthermore provides constructs for the definition and usage of *constraints*, *requirements*, and *allocations*. Note that each of the above definitions also includes attributes and constraints that are inherited to instances and then allow the user the analysis of concrete use cases with concrete values as described in Sec. IV. Furthermore, we define calculations in Sec. IV that can be used in this context.

Fig. 2 shows an example: the part type *Ethernet* is defined with the attributes *dataRate* which has a concrete value, and *length* where the value is constrained to a range. A part *myCable* is instantiated, which inherits from the definition *Ethernet* and redefines the attribute *length* with a concrete value.

```
part def Ethernet{
    attribute dataRate: Rate [MHz]=10.0 [MHz];
    attribute length: Length(0..100) [m].
}
part myCable :Ethernet{
    attribute length: Length [m] = 15.0 [m].
}
```

Fig. 2. SysML v2 textual example for the definition of types.

Allocations introduce a 1:1 relationship between features. The follow the same scheme of definition and usage as parts, constraints, and requirements. Allocation definitions describe constraints on the related element's type; usages relate to two concrete features.

Allocations can be understood as an allocation table, which can be seen in Table I. The corresponding SysML v2 code is shown in Fig. 3. The *allocation definition*, in the beginning, defines the relation *executedBy*, which connects a part of type Software with a part of type ECU. Then, the *allocation* is used with the concrete example of the table with the *allocation* a1.

TABLE I
EXAMPLE OF AN ALLOCATION TABLE

Software	ECU
SpeedMeasureSW	ZonalFrontECU
AbsSoftware	CentralECU

```
allocation def ExecutedBy {
   end part software: Software;
   end part computeUnit: ECU;
}
allocation a1 : ExecutedBy
    allocate SpeeadMeasureSW to ZonalFrontECU;
```

Fig. 3. SysML v2 textual example for the allocation of Table I

Fig. 4 gives an overview of the architecture of definitions provided by the OpenCar modeling framework. To support the specification, the framework provides requirements- and constraints definitions, which are shown on the left side of this figure. Features and functions in the logical architecture are part definitions in line with ISO26262. The allocation definition *satisfiedBy* connects the defined requirements to defined features. The allocation *getDataFrom* introduces the data flow in Logical architecture.

For modeling the view *technical architecture*, OpenCar defines the allocation *implementedBy* to software and/or hardware components. Hardware components are further classified into compute Units (CU), Sensor, Actuator, or Network (e.g. Wire) parts. The allocation *executed by* describes the mapping of Software a compute unit (CU).

The view *topology* describes the concrete locations of components in a vehicle. To define the topology, OpenCar provides definitions of locations in a 3-dimensional coordinate system and defines building spaces. The allocation *locatedIn*, relates hardware elements with a corresponding building space. These locations permit estimating environmental loads, e.g., for deriving mission profiles.

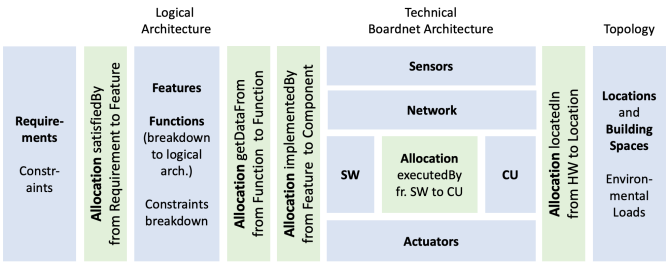

Fig. 4. OpenCar framework: Classes and allocation definitions

Furthermore, all parts of the technical architecture have a location field, where the location is stored. This location is undefined in the definition.

Instances of the parts typed by the definitions ECU, Sensor, Actor get a location in a 3D coordinate system (see Fig. 5) and building space. A building space divides the vehicle into different areas, which are connected, for example, the engine compartment or the trunk. Cables can be defined as an association from part to part with an instance of the cable. This wire can add a simple latency model depending on MAC protocols.

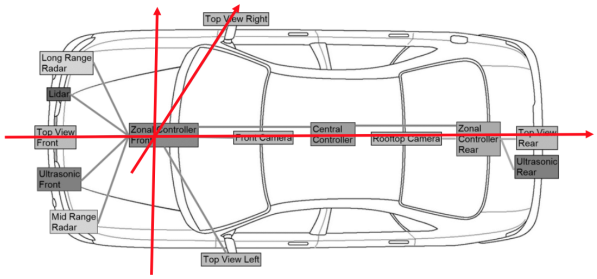

Fig. 5. Coordinate System of the car

IV. OpenCar analysis details

We have implemented a SysML v2 model for early analysis before a concrete model is available. This model consists of the following packages:

- *topology*: Defines location in a 3D coordinate system and the connection between the two locations
- *installationSpaces*: divides the vehicle into subspaces with environmental loads
- *sensors*: defines available sensors and their attributes
- *controllers*: a generic processor model is defined for the delay estimation of controllers
- *features*: defines features that can be mapped to controllers
- *network*: a generic connection model is defined, which uses PHY and MAC layer protocols for the estimation of latency and throughput and defines concrete wires
- *safety*: models reliability and ASIL level estimation methods including ASIL decomposition

We will describe some parts of the model in the following sections

A. Analysis of cable tree

As we described in [4], a method for analyzing the cable tree has been developed, including the length, weight, and costs of the cable tree. We have transformed this approach, written in the SysMD language into the common standard SysML v2. The model is also included in our OpenCar model.

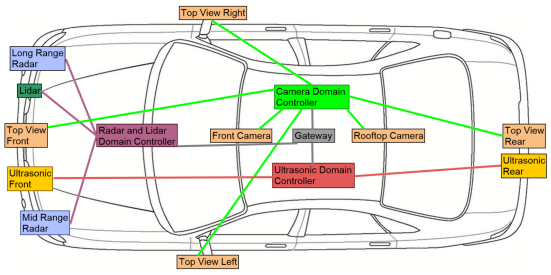

Fig. 6. Zonal architecture of the car model

We have created two basic vehicle architectures, the zonal (see Fig. 6) and the domain-centralized architecture (see Fig.

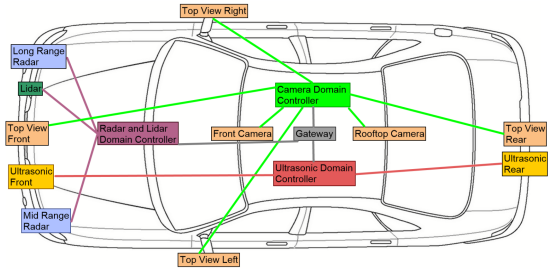

Fig. 7. Domain-centralized architecture of the car model

7). The zonal architecture consists of a central controller and two zonal controllers, in the front and the back of the car. The domain architecture consists of a central gateway connecting the three domain controllers.

```
// A location in the boardnet coordinate system
class Location {
  attribute position: Length(-1.5..6,-1.5..1.5,-0.5..4.0)[m];
  attribute plugCosts: Currency(1..2) [€];
}

// Connection from Location to Location (incl. PHY, MAC);
part def Connection {
  attribute factor: Quantity(1..2) = 1.4;
  attribute wireLength: Length(0..1000) [m] =
    cityBlockDistance(Start::position,End::position)*factor;
  attribute latency: Quantity(0..1000000) [µs] =
    (kind::dataPerFrame+kind::overheadPerFrame)/kind::dataRate;
  feature Start: Location;
  feature End:   Location;
  part kind:  openCar::network::Wire;
}
```

Fig. 8. SysML v2 model of Location and Connection with estimations.

For the analysis of this model, we have modeled the different *Sensors* and *Wires* inside our OpenCar model. The *Sensors* have a *Location* stored as 3D coordinates and the *Wires* are stored as a *Connection* between a *Sensor* and a *CU* (see Fig. 8). Each *Wire* has a defined data rate, cost per meter, specific weight, data size per frame, and overhead per frame. One wire to the midrange radar is modeled as optional. Such variants are necessary because different countries have varying regulations for vehicles compared to others. We have used the city block distance [19] as an approximation of the wire length because wires typically do not follow a direct path between components.

Fixed values for the positions of all components were applied to obtain the calculation results. These values are exemplary and have no real values. The results of these calculations are shown in Table II. It can be seen, that the cable length and mass for the domain architecture are much higher compared to the zonal architecture. The effect of using ranges for the length in the domain architecture and the variant in the zonal architecture are also visible in the results.

B. Latency Analysis

The analysis of the latency is another aspect, which can be treated by our model. The latency is defined along a given path, for example, sensor - cable - ECU - cable - actuator. The latency of a cable is estimated by the MAC layer protocol with the following formula:

TABLE II
ZONAL VERSUS DOMAIN-CENTRALIZED ARCHITECTURE.

	Domain Architecture	Zonal Architecture
Cable tree length	41.58 .. 45.63 m	35.14 m
Cable tree mass	0.628 .. 0.848 kg	0.526 .. 0.695 kg
Total cost	50.82 .. 55.84 €	52.19 €

$$latency = \frac{dataPerFrame + overheadPerFrame}{dataRate} \quad (1)$$

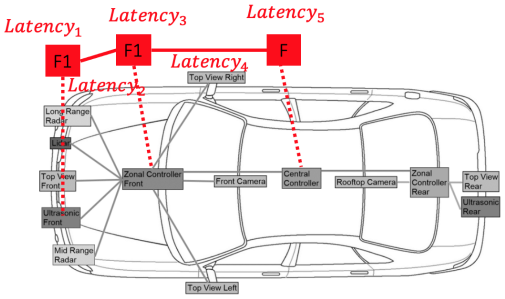

Fig. 9. Latency along a given path

All these values are already stored in the sensor file of our OpenCar model. The latency of the ECU can be calculated using a load model with the allocated features and the number of instructions per schedule. The performance of the processor should also be considered with the instructions per cycle and the clock frequency. The average and worst case must be considered. The latency of the sensor is estimated by the time to create a packet. To calculate the whole latency, all these values must be summed up at a given path. An example can be seen in Fig. 9.

C. Safety Analysis

In safety considerations, there are three main points we want to address: reliability metrics, ASIL calculation, and ASIL decomposition concerning reliability metrics. Regarding the reliability metrics, it is important to address them within the context of ISO26262, focusing on both standard and non-standard indicators, and systematically outlining them in our own SysMD language. The main metrics mentioned in ISO26262-5 [20] and ISO26262-9 [21] are single-point fault metric (SPFM), latent fault metric (LFM), as well as residual fault metric (RFM) and multiple-point fault metric (MPFM), but to ensure comprehensive coverage, additional essential reliability indicators are considered and calculated, such as diagnostic coverage (DC), residual fault probability (RFP) and safe fault fraction (SFF).

ISO26262 differentiates among various types of failure rates, namely failure rates of single-point faults, multiple-point faults, residual faults, as well as safe faults. Assuming all of them are constant, adding them up yields a total failure rate, used throughout the metrics' calculation.

$$\lambda_{total} = \lambda_{SPF} + \lambda_{RF} + \lambda_{MPF} + \lambda_S \quad (2)$$

Safe Fault Fraction (SFF), a metric mentioned in ISO26262-5, is the proportion of faults that are either detected or do not lead to the violation of the safety goal, hence ensuring that the system can manage faults safely, which is of high importance for higher ASILs [20]. The implementation of the metric may be seen in Fig. 10.

$$SFF = (\frac{\lambda_{safe} + \lambda_{detected}}{\lambda_{total}}) \tag{3}$$

```
calc def SFFCalc {
    in attribute lambdaS: Real;
    in attribute lambdaDet: Real;
    in attribute lambdaTot: Real;
    return SFF: Real [%] =
          ((lambdaS*lambdaDet)/lambdaTot);
}
```

Fig. 10. Calculating Safe Fault Fraction using SysML v2

Non-ISO-related metrics, such as availability, reliability, and probability of failure, indirectly impact ASIL by influencing the system's behavior during failures, thus contributing to the overall safety, hence are also calculated within the safety analysis. Namely, reliability itself is the probability that a system or a component performs its required function(s) without failure over a specific period of time [22], hence it is calculated and implemented using an exponential function.

$$R(t) = e^{-\lambda t} \tag{4}$$

```
calc def ReliabilityCalc {
    in attribute lambda: Real;
    in attribute time: Real;
    return R: Real = exp(-lambda*time);
}
```

Fig. 11. Calculating Reliability using SysML v2

Another important aspect of safety analysis is the ASIL level analysis, which uses three metrics to classify the risk of a system in levels from ASIL A (lowest risk) to ASIL D (highest risk) or quality management (QM). The metrics used for the classification are severity from SO (no injuries) to S3 (fatal injuries), exposure from E0 (very unlikely) to E4 (highly possible), and controllability from C0 (easy to control) to C3 (hard to control or uncontrollable). Fig. 12 shows how to use the metrics to get the ASIL level of a system.

OpenCar implements the ASIL calculation in SysML v2 which can be seen in Fig. 13. Integers represent the different metrics and ASIL levels. The implementation is based on Schacht's [2] usage of integers for the ASIL levels. This allows us to use e.g. integer ranges and in particular limits the use of discrete enumerations that put load on the (slower) discrete solver part that handles enumerations.

As reliability metrics play a pivotal role in determining ASIL levels, they are to be integrated with ASIL levels, by specifying how different metrics impact ASIL. A comprehensive review of target values for different metrics across different ASILs can be seen in Table III. As shown, each

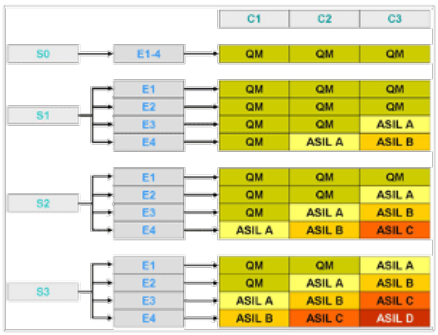

Fig. 12. Table of ASIL levels [2]

```
calc def calcASIL{
    in attribute severity : Integer(0..3);
    in attribute exposure : Integer(0..4);
    in attribute controllability: Integer(0..3);
    attribute sum: Integer = severity + exposure
        + controllability;
    //special case for C0 and S0
    attribute sumAdapted : Integer = if severity == 0
        or controllability == 0 ? 0 else sum;
    //0->QM, 1->ASIL A .. 4->ASIL D
    return result: Integer = max(sum-6,0);
}
```

Fig. 13. Calculate ASIL level using SysML v2

metric has a specific target value for ASIL A, B, C, and D, hence serving as a crucial reference point for understanding and defining how the metrics relate to ASIL levels, and properly calculate the required ASIL.

TABLE III
METRIC TARGETS FOR DIFFERENT ASILS [20], [21]

Metric	ASIL A	ASIL B	ASIL C	ASIL D
SPFM	Not specified	$\geq 90\%$	$\geq 97\%$	$\geq 99\%$
LFM	Not specified	$\geq 60\%$	$\geq 80\%$	$\geq 90\%$
DC	$\geq 60\%$	$\geq 90\%$	$\geq 99\%$	$\geq 99.9\%$
SFF	$\geq 60\%$	$\geq 90\%$	$\geq 99\%$	$\geq 99\%$
Availability	$\geq 99\%$	$\geq 99.9\%$	$\geq 99.99\%$	$\geq 99.999\%$

Integrating ASIL decomposition involves breaking down the overall safety goal into smaller, more manageable safety subgoals, as shown in Fig. 14, allowing for a more targeted, efficient, and easier approach to achieve safety objectives, e.g. a ASIL D system may be decomposed into two redundant ASIL B systems, contributing to the total safety and reliability of the system as per ISO26262 standard. Only a small part for the selection of the correct value of redundant subsystems needs to stay at the high ASIL level (level D in the example above).

The implementation of the ASIL decomposition is shown in Fig. 15. The function uses the ASIL level of two parts and a boolean if they are redundant as inputs. For the not redundant case, the overall ASIL level cannot be higher than the minimum ASIL level of the components. Otherwise, the ASIL levels can be summed up, but the highest ASIL level D (represented as 4) must be considered.

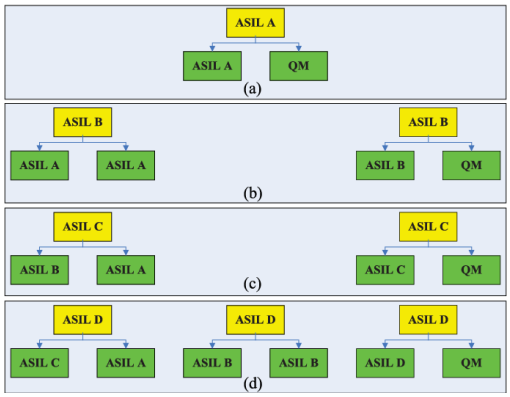

Fig. 14. ASIL Decomposition [23]

```
calc def calculateASILDecomposition{
    in attribute part1: Integer(0..4);
    in attribute part2: Integer(0..4);
    in attribute isRedundant: Boolean;
    attribute redundantLevel: Integer
        = min(part1+part2, 4);
    attribute notredundantLevel: Integer
        = min(part1, part2);

    return result: Integer(0..4) = if isRedundant ?
        redundantLevel else notredundantLevel.
}
```

Fig. 15. Calculation of ASIL decomposition using SysML v2.

V. Conclusion and Outlook

OpenCar is a SysML v2 modeling framework for automotive applications. It supports collaboration along the value chain by offering tools for modeling, analyzing, and optimizing vehicle boardnet architectures. Critical metrics such as cable mass, cost, and latencies are essential for vehicle architecture optimization. The additional features for safety and reliability analyses in line with ISO26262 ensure compliance with today's automotive safety standards. We in particular expect that collaborations between OEMs and the suppliers along the value chain can be enhanced by the modeling framework: SysMLv2 models and a comprehensive modeling framework simply can be more accurate and more meaningful compared with Excel sheets. Furthermore, the calculations as defined can easily be re-used and adapted.

The SysML v2 modeling framework with OpenCar as of now provides many calculation functions that as of now are often done using tools like Excel. As of now, the defined calculations and constraints are handled by a constraint propagation solver. While the models themselves mostly follow the SysML standard, the (range-and set-based) semantics of the calculations and the constraint solving in the tool are not yet standardized. However, shortly (Autumn 2024) the submission of an RFC to the OMG is expected.

References

[1] J. Klaus-Wagenbrenner, "Zonal EE Architecture: Towards a Fully Automotive Ethernet–Based Vehicle Infrastructure." https://standards.ieee.org/wp-content/uploads/import/documents/other/eipatd-presentations/2019/D1-04_KLAUS-Zonal_EE_Architecture.pdf. [Online; visited 11-July-2023].

[2] J. Schacht, "Funktionale Sicherheit (FuSi) – die ASIL-Klassifikation." https://www.i-q.de/iso-26262/fusi-asil-klassifikationen, 4 2016. [Online; visited 28-June-2024].

[3] M. Berk, O. Schubert, H.-M. Kroll, B. Buschardt, and D. Straub, "Exploiting redundancy for reliability analysis of sensor perception in automated driving vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5073–5085, 2020.

[4] S. Post and C. Grimm, "Co-design of automotive boardnet topology and architecture," in *DVCon Europe 2023*, pp. 42–49, 2023. https://ieeexplore.ieee.org/document/10461369.

[5] A. Ratzke, S. Post, J. Koch, and C. Grimm, "Constructive model analysis of SysMLv2 models by constraint propagation," in *System of Systems Engineering Conference 2024*, IEEE, Jun 2024.

[6] H. G. C. Góngora, T. Gaudré, and S. Tucci-Piergiovanni, "Towards an architectural design framework for automotive systems development," in *Complex Systems Design & Management* (M. Aiguier, Y. Caseau, D. Krob, and A. Rauzy, eds.), (Berlin, Heidelberg), pp. 241–258, Springer Berlin Heidelberg, 2013.

[7] R. Blank, "Boardnetzprozess 4.0: Zeit für einen Paradigmenwechsel." https://www.elektroniknet.de/automotive/software-tools/zeit-fuer-einen-paradigmenwechsel.126606.html, 2016.

[8] M. Soltani and E. Knauss, "Challenges of requirements engineering in AUTOSAR ecosystems," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pp. 294–295, 2015.

[9] P. Pelliccione, E. Knauss, R. Heldal, S. Magnus Ågren, P. Mallozzi, A. Alminger, and D. Borgentun, "Automotive architecture framework: The experience of Volvo Cars," *Journal of Systems Architecture*, vol. 77, pp. 83–100, 2017.

[10] G. Richardson, J. Shultz, P. Stevens, and J. Wilson, "Development of an electrical data exchange interface based on step ap212," *SAE Transactions*, vol. 114, pp. 572–582, 2005.

[11] OMG, "Systems Modeling Language (SysML v2), Release 05/2022." https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/2-OMG_Systems_Modeling_Language.pdf.

[12] OMG, "Kernel Modeling Language (KerML), Release 05/2022." https://github.com/Systems-Modeling/SysML-v2-Release/blob/master/doc/1-Kernel_Modeling_Language.pdf.

[13] H. Martin, Z. Ma, C. Schmittner, B. Winkler, M. Krammer, D. Schneider, T. Amorim, G. Macher, and C. Kreiner, "Combined automotive safety and security pattern engineering approach," *Reliability Engineering & System Safety*, vol. 198, p. 106773, 2020.

[14] S. Yandika, C. Baron, C. Bonnard, L. Pahun, L. Grenier, and P. Esteban, "Investigating the use of a model-based approach to assess automotive embedded software safety," in *13th International Conference on Modeling, Optimization and Simulation (MOSIM20)*, 09 2020.

[15] Y. Zhen-Hua, M. Ling-Xu, and H. Jun-Nan, "Application of preevision software to realize vehicle functional safety development," in *2021 5th International Conference on Vision, Image and Signal Processing (ICVISP)*, pp. 55–60, 2021.

[16] B. Zheng, H. Liang, Q. Zhu, H. Yu, and C.-W. Lin, "Next generation automotive architecture modeling and exploration for autonomous driving," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 53–58, 2016.

[17] B. M. Randles, I. V. Pasquetto, M. S. Golshan, and C. L. Borgman, "Using the jupyter notebook as a tool for open science: An empirical study," in *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pp. 1–2, 2017.

[18] M. Broy, M. Gleirscher, S. Merenda, D. Wild, P. Kluge, and W. Krenzer, "Toward a holistic and standardized automotive architecture description," *Computer*, vol. 42, no. 12, pp. 98–101, 2009.

[19] R. A. Melter, "Some characterizations of city block distance," *Pattern Recognition Letters*, vol. 6, no. 4, pp. 235–240, 1987.

[20] International Organization for Standardization, "ISO 26262-5:2018: Road vehicles - functional safety; part 5: Product development at the hardware level," 2018. Accessed: 2024-06-28.

[21] International Organization for Standardization, "ISO 26262-9:2018: Road vehicles - functional safety; part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses," 2018. Accessed: 2024-06-28.

[22] A. Bruton, J. H. Conway, and S. T. Holgate, "Reliability: What is it, and how is it measured?," *Physiotherapy*, vol. 86, no. 2, pp. 94–99, 2000.

[23] G. Xie, Y. Chen, Y. Liu, R. Li, and K. Li, "Minimizing development cost with reliability goal for automotive functional safety during design phase," *IEEE Transactions on Reliability*, vol. 67, pp. 196–211, 2018.