Developing a Method for an Optimized Static Vehicle Function Distribution

Jan Ruhnau*‡§ and Steffen Becker§
*Mercedes-Benz AG, Mercedesstr. 120, 70372 Stuttgart, Germany
‡Graduate School of Excellence advanced Manufacturing Engineering, Nobelstraße 12, 70569 Stuttgart, Germany
§University of Stuttgart, Keplerstr. 7, 70174 Stuttgart, Germany
{jan.ruhnau@mercedes-benz.com, steffen.becker@iste.uni-stuttgart.de}

***Abstract*—The design of automotive electric/electronic (E/E) architectures has become increasingly complex due to the integration of numerous software-controlled functions. To address this complexity, we present a generic method for developing and implementing a vehicle function distribution optimization process. Our approach involves defining the design space, gathering relevant information from various sources, solving the optimization problem, and displaying the best configurations. Two processes are described for firstly developing, and successively using a method for the optimization. The resulting method is validated through two use-cases from the series development of current Mercedes vehicles. Validation runs show that our method can effectively optimize function distributions, reduce overhead costs, and improve system quality criteria in general, therefore providing a solid foundation for enhancing the design and development process of automotive E/E architectures.**

***Index Terms*—static vehicle function distribution, E/E architecture optimization**

I. Introduction

When a new automotive vehicle is developed, one of the most important aspects nowadays is the electric/electronic (E/E) architecture. This is due to many new customer functions implemented in software. However, designing the functional and hardware architecture comprising the E/E architecture becomes an increasingly complex task. Since much of the architecture design process is still handled manually, we see a benefit in implementing a design method that at least partly automates this process. This would allow checking many of the constraints present in the planning concurrently, as well as increase the number of configurations that can be evaluated.

In this work we present a process for developing and implementing a vehicle function distribution method. This allows architects to (i) define the design space and settings, (ii) manually or automatically gather relevant information from the data model in the environment, (iii) solve the optimization problem and (iv) display the best found options in a list.

In order to implement and use the optimization method for static vehicle function distributions described in this work we provide *Business Process Model and Notation* (BPMN) diagrams that show the different tasks that need to be fulfilled by participating parties. Furthermore, we also showcase our implementation and validation of the method with the execution in two use-cases from the series development of a new vehicle architecture, and highlight the feedback of the architects.

II. Related Work

Over the past two decades, a variety of different architecture optimization approaches were introduced to tackle the growing complexity of the designed systems. Approaches exist for either generic use-cases, but also specifically designed for automotive architectures.

Koziolek and Reussner [1] proposed a flexible and extensible framework for optimizing component-based system (CBS) models across multiple quality properties and degrees of freedom. The framework utilizes a generic metamodel to describe degrees of freedom for any CBS metamodel, allowing for the automatic derivation of optimization problems and the application of multi-objective metaheuristic optimization techniques, such as evolutionary algorithms. The authors demonstrate the feasibility of their approach through the implementation of a CBS-metamodel-agnostic transformation and discuss plans to integrate this generic transformation into their existing optimization tool, PerOpteryx, to support various CBS metamodels.

Kugele et al. [2] presented a generic framework for model-based optimization of automotive E/E-architectures using a domain-specific language called AAOL (Automotive Architecture Optimization Language). Their approach addresses the deployment problem by introducing a holistic architectural model that supports seamless model-based development from requirements management to deployment. The AAOL language allows for the expression of a wide range of deployment-relevant problems and supports multi-objective evolutionary algorithms (MOEA) for optimization. The feasibility of the approach is demonstrated through a case study from the automotive domain.

Arcelli et al. [3] introduced EASIER, an Evolutionary Approach for multi-objective Software archItecturE Refactoring, aimed at optimizing software architecture refactoring based on performance metrics and the intensity of changes. EASIER employs evolutionary algorithms to generate refactoring sequences that improve architectural performance while minimizing the distance from the initial architecture and reducing performance antipatterns. The approach integrates performance analysis and refactoring within the Æmilia ADL environment, leveraging a custom NSGA-II algorithm to search for optimal solutions. The authors demonstrate the

effectiveness of EASIER through a case study, showcasing its ability to produce Pareto-optimal solutions that balance performance improvements and architectural changes.

Kugele et al. [4] also proposed a methodology for analyzing hardware resources and synthesizing automotive service-oriented architectures based on platform-independent service models. The approach applies design space exploration and simulation to analyze and synthesize deployment configurations, mapping services to hardware resources at an early development stage. These configurations are then refined into AUTOSAR Adaptive software architecture models, providing the necessary input for subsequent implementation processes. The authors demonstrate the feasibility of their approach through deployment configurations optimized for computing resources and simulation results that meet runtime requirements, supported by a prototype software toolchain.

Ni et al. [5] proposed MORE (Multi-objective performance Optimisation based on Randomised search rulEs), a novel approach for optimizing software performance at the architecture level. MORE integrates practical performance improvement knowledge with search-based software engineering techniques. It introduces randomised search rules (MORE-R) that do not require predefined thresholds or improvement amplitudes, making them easier for software architects to use and providing parameter-free explanations. The approach employs a multi-objective evolutionary algorithm (MORE-EA) to efficiently solve MORE-P. Experiments demonstrate that MORE achieves higher quality and more explicable solutions than state-of-the-art techniques, showing the benefits of combining search-based approaches with practical knowledge.

An in-depth literature review on the topic of software architecture optimization methods was published by Aleti et al. [6]. The review also highlights challenges stemming from the absence of a standardized terminology in software architecture design. This lack of a consistent taxonomy hampers research and communication within these groups.

Our work differs from the related approaches in several key aspects. While previous methods focus on specific frameworks and domain-specific languages for optimizing automotive and component-based system architectures, our approach is more generic and flexible. We provide a method that includes two processes for developing and using an optimization method tailored to the specific needs of automotive E/E architectures. Unlike previous work, our method is not constrained by preexisting frameworks or languages. Instead, we offer a greenfield approach that adapts the problem description and optimization algorithm based on the criteria collected by the architects. This ensures broad applicability and practical use without external dependencies, however, it also does not restrict the usage of these frameworks. Additionally, our approach integrates the entire process, from defining the design space over solution aspects to displaying the best configurations. This end-to-end process provides a holistic solution that can easily be adapted and validated in real-world scenarios, demonstrating that even straightforward implementations can effectively address the complexities of automotive E/E architecture design.

III. Preliminaries

A. Ontology

To explain the most important terminology in the context of 'function distribution', we reference to earlier work introducing an ontology [7]. One of its main aspects is that each valid function distribution consists of a deployment and an allocation, which in term then consists of many single deployment- and allocation-mappings. An allocation-mapping then describes on which execution node each software component (SWC) is planned to execute on, whereas the deployment-mapping describes where the artifact implementing the SWC is finally installed (hence deployed) on. This paper mostly focuses on electronic control units (ECUs) as one type of execution. The construct of SWCs realizes the sub-functions and therefore also the customer functions.

B. Use-Cases

There are several significant use-cases in enhancing the design and development process of vehicle E/E architectures. Primarily, it facilitates the integration of the functional architecture (encompassing software components and communication) with the hardware architecture (including ECUs and buses). This integration accelerates the design process by enabling the concurrent testing of multiple constraints. Furthermore, our method allows the exploration of an expanded solution space by evaluating a very high number of distinct configurations spanned by the defined degrees of freedom, which cannot be fully explored by manual evaluation. This allows to rapidly validate or disprove new hypothetical configuration. For example, the method supports experimentation with various placements of software components or adjustments in hardware elements, allowing for the assessment of potential impacts and feasibility of different configurations. These possibilities in quick hypothesis testing then accelerate the entire decision-making process. All in all, our method provides enhanced flexibility in the design process, particularly in challenging scenarios. It can accommodate last-minute changes, adapt to hardware supplier modifications, and foster increased competition among hardware suppliers.

IV. General Description of the Development of the Method

This section outlines the development of a method for static vehicle function distribution using a BPMN diagram (Figure 1). The diagram includes three parties: functional architects, method developers, and domain experts, each represented by a swim lane. Functional architects, responsible for placing functions on vehicle hardware, initiate the process as they benefit most from automation. Method developers implement optimization algorithms, while domain experts provide details and criteria for these algorithms.

To ensure a valid and good quality function distribution, specific requirements must be met. After process initiation, criteria for function distribution are collected, such as resource requirements or latencies. Architects decide which criteria are important for their use-cases and classify them as constraints

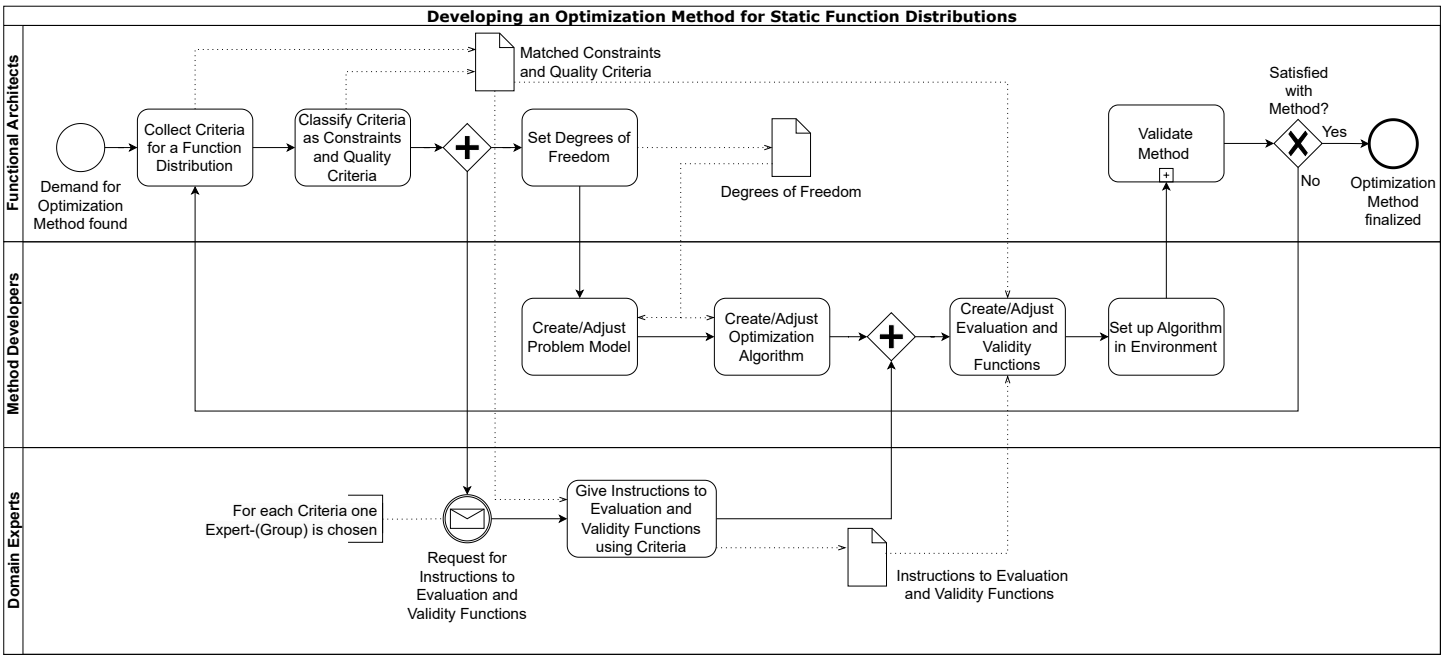

Fig. 1. BPMN diagram showing the development of a method for static function distributions

(necessary for valid distributions) or quality criteria (used for evaluation), or both. This list forms the basis for algorithm implementation. Architects also set the algorithm's degrees of freedom, such as reallocation of SWCs or adaptability of ECUs or buses. Domain experts advise on designing algorithms to examine the validity or quality of criteria, which is later used for implementing according functions.

With the degrees of freedom defined, method developers create a problem model to hold data relevant to constraints and quality criteria for optimization. The actual implementation of these optimization algorithms follows next. Discrete optimization approaches are suitable, including heuristics like tabu-search, simulated annealing, or evolutionary algorithms. Depending on constraints and degrees of freedom, other algorithms for constraint programming or SAT problems may be feasible. Using criteria from architects and input from domain experts, evaluation and validity functions are integrated into optimization algorithms. These must then be set up in a suitable environment, capable of hosting the algorithm, handling input data, and outputting optimization results. It should be accessible and usable by functional architects, the main users. This may include features to assist in setting optimization options and other steps of the method usage.

After finalizing the prototype, the next step is to validate its usefulness by applying it to intended use-cases, as described in the following section and in Figure 2. Validation may reveal improvements, leading to a potential iterative refinement.

V. General Description of Using the Method

After developing the automotive vehicle function distribution method, its usage is modeled in a BPMN graph (Figure 2). Functional architects are key actors, initiating and controlling the process. Component and system managers may also trigger the process due to the need for function redistribution on their hardware components or systems. Since they are the experts of the functions they further serve as data source for the process and also take part in the final evaluation of the solutions.

Upon identifying the need for function (re-)distribution, architects gather data about the problem, including lists of ECUs, buses, SWCs, and their communication. This can be challenging due to the interconnected nature of modern automotive functional architecture. Automatic model checking helps manage this complexity by identifying and adding relevant information. Depending on the data situation and criteria requirements, component and system managers may need to provide additional information, such as ECU resource data (e.g., available memory), essential for optimization. Architects enter ECUs, SWCs, buses, and communication data into the method's environment. Once additional data from managers is available, it completes the local data model.

Architects then apply settings for the optimization algorithm, which may include time constraints and prioritization of optimization criteria. The optimization runs are then started, and the algorithm proposes viable distributions. The final stage involves evaluating these distributions with all participants. If one or more solutions meet all needs, architects choose a distribution, completing the process. If no solutions are satisfactory, the process can restart with new data or adjusted optimization parameters to find better solutions.

VI. Prototypical Implementation of the Method

Following the BPMN diagram, we implemented a prototype for optimizing static function distributions.

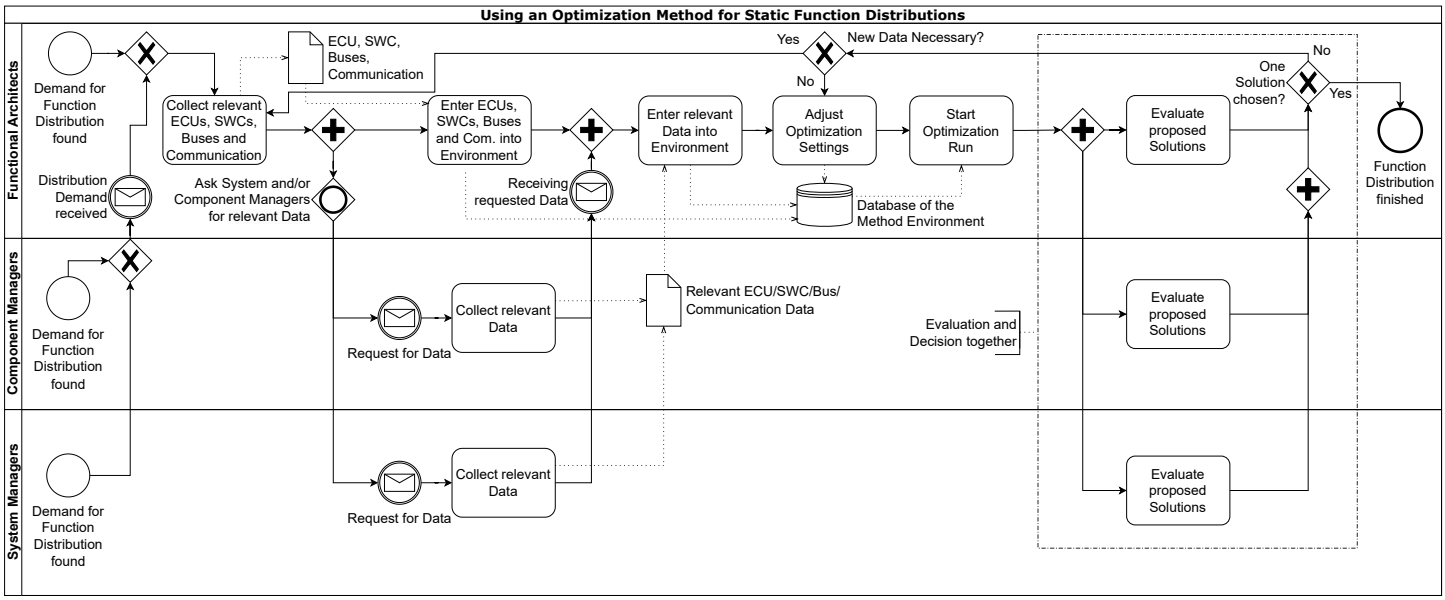

Fig. 2. BPMN diagram showing the possible process of using a method for static function distributions

A. Classified Criteria to a Valid Function Distribution

In the beginning, the functional architects collected criteria for function distributions and classified them as constraints (C) or quality criteria (Q). Table I provides an overview.

TABLE I
A LIST OF CRITERIA, CLASSIFIED BY 'C' FOR CONSTRAINTS AND 'Q' FOR QUALITY CRITERIA.

Criteria	**Classification**
Unique Assignment	C
Functional Safety	C, Q
ECU Resources	C
Certifications	C, Q
Startup Time	C, Q
Timing	C
Scheduling on ECUs and Bus Systems	C
Bus and Gateway Utilization	C, Q
Real-time	C
Security	C
Amount of Updates/Upgrades and Releases	C, Q
Maximum Operating Hours	C
Hardware Dependency	C
Designer Preference	C
Probability and Mean Time of Failure	C, Q
Power Consumption	Q
Utilization of ECUs	Q
Cost	Q

B. Degrees of Freedom

In the development of automotive E/E architectures, hardware and functional architectures are closely interdependent. However, for the inital prototype we decided focusing solely on allocating predefined software components to hardware, hence assuming ECUs and buses are fixed at the beginning.

C. Problem Model

We implemented the problem and allocation algorithms in Java to integrate with the E/E development tool PREEvision. For data handling and solving the optimization we created an extendable 'function allocation problem' class. Its scope is based on chosen degrees of freedom and criteria. The problem model includes attributes like startup times of ECUs and SWCs (`int[] S_st, int[] C_st`), multiple resources of SWCs and ECUs (`int[][] S_res, int[][] C_res`), bandwidth in-between SWCs (`int[][] S_bw`), connecting buses in between ECUs (`int[][] C_b`) and so on. Further Key structures include the allocation-constraints matrix (`boolean[][] AC`) and pre-allocation-vector (`int[] Z`), which show if a SWC-ECU pair is allowed or even fixed from the beginning respectively.

D. Optimization

We implemented two approaches for the optimization algorithms: random search and evolutionary algorithm. Both can be considered as heuristics for solving the optimization problem by creating solution configurations in the beginning. More specifically, for a single solution they are mapping all SWCs to ECUs, check if constraints are met by executing a number of validity functions and finally evaluate valid solutions with the evaluation functions. For the random search this process is simply repeated over and over again for a fixed number of steps. The evolutionary algorithm also creates random allocations, however after an initial population of solutions has been found, it creates further descendants between several good solutions by executing crossover functions in between them and also creates more mutations, and culls over-population. The process continues until a stopping criterion is met.

E. Validity Checks

The algorithm verifies the solutions validity based on constraints set by architects, including:

- Unique Assignment: Ensures each SWC is assigned to an ECU.
- Functional Safety: Compares SWC and ECU ASIL (Automotive Safety Integration Level).
- ECU Resources: Checks if ECU resources meet SWC requirements.
- Startup Time: Compares SWC and ECU startup times.
- Bus Utilization: Evaluates bus workload against thresholds.
- Latency: Checks communication latencies between several SWCs.
- Certifications: Ensures SWCs requiring certifications are on certified ECUs.
- Real-time: Ensures real-time SWCs are on real-time capable ECUs.
- Designer Preference: Checks pre-allocation vector and allocation constraints.

F. Evaluation of Solutions

Solutions are evaluated and assigned a cost value based on evaluation functions. These can be of arbitrary form or complexity and therefore also be non-linear and non-continuous functions. They are representing the optimization criteria that are wanted in an optimal solution. Criteria implemented in our prototype includes:

- Used ECUs: More ECUs increase costs.
- Bus communication: Bandwidth between ECUs adds costs (depending on bus type).
- Surpassing Bus Communication Thresholds: Exceeding bus utilization thresholds adds costs (linearly or exponentially).
- Certification on ECUs: SWCs requiring certification add related costs.
- Inclusion of SWCs without need of certification: Adds costs for non-certified SWCs on certified ECU.
- ASIL overhead: Adds costs for lower ASIL SWCs on higher ASIL ECUs.
- ASIL hardware costs: Adds costs based on ECU ASIL.
- Complexity: Increases costs based on the number of signals on buses.

G. Setting up the Algorithm in an Environment

As already described above, the method was implemented in the E/E development environment PREEvision. Data already present in this tool includes the topology, communication data and software architecture with all components. This is automatically combined with further input from architects to create an optimization object. For example, the user of the method is able to choose a subset of SWCs and ECUs.

The algorithm, is set up to extend the chosen problem with adjacent communication without increasing the solution space. E.g. when five SWCs should be redistributed onto three ECUs, however important sensor input comes from additional SWCs that should not be redistributed, an extended problem is created that includes these additional SWCs with fixed pre-allocation vector entries. This allows to include important boundary properties without increasing the solution space. The method was implemented in a metric diagram, which is the PREEvision way of including custom java code.

Resulting function distributions are finally displayed in console outputs as well as some tables, showing SWC-ECU allocations and costs for each evaluation criterion. Architects can choose to display all SWCs, only input SWCs, or only non-pre-allocated SWCs. The results from the algorithm are combined with data from the environment to show human-readable names instead of only numbers.

H. Categorization of Implementation

For the cagegorization of the implementation we are relying on the taxonomy given by Aleti et al. [6]. Our created implementation of the generic method is designed for problems of embedded systems in the design time phase and can be classified as multi objective optimization due to its many quality criteria and constraints described in table I. We focused on the degree of freedom of the allocation of SWC to ECUs by using a custom architecture model. Due to this customization, the evaluation functions can be very simple additive functions up to any conceivable nonlinear mathematical functions. Constraints are handled via penalty functions as well as prohibition of specific solutions. Our solution algorithm cannot guarantee the exact solution, therefore finds approximations by using metaheuristics. For the validation we used industrial case studies.

VII. Validation with First Use-Cases

The validation of the method was conducted using two specific use-cases from the series development of current Mercedes vehicles. Below, we introduce these use-cases, discuss how they benefit from the method, and list suggestions for improvement. Due to confidentiality, details of the systems are kept vague.

A. Gathering of Relevant Data

Before optimization, relevant data was gathered from multiple sources, including the PREEvision data model and inputs from experts. PREEvision served as the source for available ECUs and SWCs, as well as communication data. Additional data required for evaluation and validity checks was collected from architects and system supervisors, including resources, startup times, certifications, and latency constraints. Customizations were made to PREEvision artifacts to hold the required information, such as domain-specific certifications and onboard diagnosis (OBD) relevance.

Furthermore, the definition of the optimization domain was required by the architects using the tool. This included the choice of the SWCs that should be optimized, as well as the options of ECUs that should be considered as potential execution nodes for them. The architects hereby had the chance to restrict certain allocation-mappings with the allocation-constraints or to pre-define others with the pre-allocation-vector. Function-chains for latency-constraints that stretch over several SWCs were also be defined.

B. Use Case 1

The first use case involves a system providing customer functions in the body and comfort domain, with high latency sensitivity but no specific ASIL requirements. The core system comprises 5 SWCs distributed over 3 ECUs, extending to 21 SWCs when considering adjacent systems via the automatic model checking. Historically, this system is allocated on an ASIL D ECU; not ideal due to its low ASIL requirements.

1) Results: The optimization algorithm proposed 5 different valid solutions, including the distribution of the main SWC onto the currently planned ECU and alternatively two other higher ECUs with ASIL B, leading to lower overhead costs. The algorithm effectively recognized and avoided solutions where the distribution lead to unsuitably high latencies.

2) Feedback of Architect: The architects were satisfied with the results, since it came to the same conclusion as their own considerations after continuous work on the same functions. However, they highlighted the need for unified data sources and the ability to switch to other hardware architecture variants for rapid prototyping and for future proofing the system. The dependency on communication data from another tool was listed as a potential weakness in new architectures.

C. Use Case 2

The second use case involves a common sub-function used by other customer functions. Its distribution constraints were focusing on startup time and complexity in technical distribution. The function, comprising a single SWC, is currently fixed on an ASIL D ECU but only requires ASIL B. The extended problem description lists 7 SWCs.

1) Results: The algorithm found 3 valid options, including the current allocation and two other higher-level ECUs with low startup times. All solutions were comparable in terms of partial costs, with minor differences in communication and ASIL overhead costs. More solutions were found to be unsuitable due to startup timing requirements and were therefore discarded.

2) Feedback of Architect: The architects of use case 2 were also pleased with the results of the method since it corresponded to their manual considerations. They were especially fond of the idea of unified evaluation of criteria throughout several distribution-problems in the entire architecture, which could impede non-technical interests in the distribution of functions. They suggested further criteria specific to this use case and emphasized the importance of providing valuable output, including reasons for ruled-out distributions and skipped checks due to lack of data input.

D. Closing Thoughts

The validation runs demonstrated the functionality and usefulness of several criteria, despite some missing data. The small solution spaces in the use-cases were easily handled by the random solutions heuristic, however also the more advanced genetic algorithm was shown to be functional. Therefore, larger solution spaces and more comprehensive data sources could greatly improve the validity of the approach.

VIII. Conclusion and Future Work

In this paper, we presented a method for developing and implementing a vehicle function distribution optimization process. Our approach integrates the entire process, from defining the design space and gathering relevant information to solving the optimization problem and displaying the best configurations. By providing a generic and flexible method, we demonstrated how a simple implementation without external dependencies can effectively address the complexities of automotive E/E architecture design.

The method was validated through two specific use-cases from the series development of current Mercedes vehicles. The results showed that our approach could successfully optimize function distributions, considering various constraints and quality criteria. The feedback from architects highlighted the practical applicability and usefulness of the method, while also pointing out areas for improvement, such as the need to adjust the hardware architecture.

In future work, further evaluation of the created prototypes is necessary to enhance the method's robustness and applicability. This includes incorporating more degrees of freedom in hardware and evaluating larger solution spaces. By expanding the degrees of freedom, we can explore a wider solution space and optimize the function distribution more effectively.

Acknowledgment

This publication is based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action.

Special thanks also go to the Ministry of Science, Research and Arts of the State of Baden-Württemberg within the sustainability support of the projects of the Exzellenzinitiative II.

References

[1] A. Koziolek and R. Reussner, "Towards a generic quality optimisation framework for component-based system models," in *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering - CBSE '11*. ACM Press, 2011.

[2] S. Kugele and G. Pucea, "Model-based optimization of automotive e/e-architectures," in *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis*, ser. CSTVA 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 18–29. [Online]. Available: https://doi.org/10.1145/2593735.2593739

[3] D. Arcelli, V. Cortellessa, M. D'Emidio, and D. D. Pompeo, "EASIER: An evolutionary approach for multi-objective software ArchItecturE refactoring." IEEE, Apr. 2018.

[4] S. Kugele, P. Obergfell, and E. Sax, "Model-based resource analysis and synthesis of service-oriented automotive software architectures," *Software and Systems Modeling*, Sep. 2021.

[5] Y. Ni, X. Du, P. Ye, L. L. Minku, X. Yao, M. Harman, and R. Xiao, "Multi-objective software performance optimisation at the architecture level using randomised search rules," *Information and Software Technology*, vol. 135, p. 106565, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921000483

[6] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 658–683, 2013.

[7] J. Ruhnau, M. Sommer, J. Henle, A. Walz, S. Becker, and E. Sax, "Ontology for vehicle function distribution," in *2023 IEEE International Systems Conference (SysCon)*, 2023, pp. 1–6.