Challenges in Automotive Hardware-Software Co-Configuration

Florian Jost[1,*], Carsten Sinz[2]

[1]Mercedes-Benz AG, Leibnitzstr. 2, Böblingen, 71032, Germany
[2]Karlsruhe University of Applied Sciences, Moltkestraße 30, 76133 Karlsruhe, Germany

Abstract
Car manufacturers offer their customers an enormous number of configuration options to individualize their vehicles. While configuration options mostly covered physical components in the past, over the last years the number of software-related options has increased immensely. Existing systems for car configuration should thus be optimized and extended to handle the shift towards more software-related features, e.g. for automatic driver assistance systems. In this article, we highlight different problems and properties combined systems of hardware-software configurations have to tackle in an automotive context.

Keywords
Automotive Configuration, Hardware-Software Co-Configuration, Future Challenges

1. Introduction

Modern car manufacturers offer a vast number of configuration options for their products. In the past, these options covered parts and functionality that were primarily based on physical components. With the ongoing electrification of cars, the rate of functionality implemented in software and thus also the variance in software is increasing [1].

But not only the number of configuration options is increasing, the same holds for the complexity of their interdependencies. Some options are mutually dependent, others are mutually exclusive. This results in an enormous configuration space with more than 10^{100} possible constructable (valid) configurations for a product line [2]. The inherent complexity of the problem challenges automotive manufactures in all stages of the product lifecycle, from development, through production, sales to after-sales. Due to the increasing importance of software in this context, hardware-software co-configurations are playing an increasingly substantial role.

In this publication, we describe upcoming or already existing problems that arise in the context of increasingly software-driven vehicles.

2. State of the Art

Classically, the possible configurations through which a car can be realized are represented by configuration options. In addition to different color finishes, these can also describe different engine designs, or optional extras such as an improved infotainment system. An existing order, i.e. a set of configuration options, can then be translated into the physical parts that are needed to manufacture the particular car instance.

Historically, with the increasing emergence of software functionality and the associated ECUs (electronic control units) in the car, the existing hardware configuration systems were adapted to also manage software configurations. But, as automotive software can have a wide variety of requirements for the installed hardware (e.g. sensors for autonomous driving systems), software configuration cannot be treated independent of the hardware configuration. However, this close connection is often not reflected in current configuration systems, where mostly software configuration is treated as a second configuration step, after the physical components have been selected and configured.

Additionally, software often comes with configurable parameters that can be set to different values. As an example, emergency call numbers differ from country to country and have to be set accordingly. The basic functionality of the software remains the same, independent of the value the parameter is set to. Thus, instead of writing software for each possible parameter setting, an ECU runs through an additional configuration step, where correct parameter values are written to the ECU (special bits get set in the programmable ROM of the ECU). The high-level software then can adapt its functionality by reading the corresponding bits in the ROM. In the au-

ConfWS'24: 26th International Workshop on Configuration, Sep 2–3, 2024, Girona, Spain
*Corresponding author.
florian_benedikt.jost@mercedes-benz.com (F. Jost); carsten.sinz@h-ka.de (C. Sinz)
0009-0006-9670-8856 (F. Jost); 0000-0001-9718-1802 (C. Sinz)

tomotive sector, this configuration step is often called variant coding [3].

Typically, in the automotive industry, configuration options are realized through Boolean variables, so-called *codes*, where each code is represented by a variable name a.k.a. identifier. An option selected by a customer is then reflected by setting the corresponding code to true. Besides codes to register customer's selections, internal codes are used, for example, there can be codes representing spatial regions and codes indicating if the car possesses left or right steering.

The set of valid configurations is described by a set of Boolean formulas (*rules*, *constraints*) that all have to be satisfied in a valid order. Such constraints can express, e.g., mutual exclusivity, as in the selection of left-hand or right-hand steering. But often constraints are much more evolved, encompassing many dozens of codes.

Example:

$$c_1 \rightarrow \neg c_2 \wedge \neg c_3$$

with codes c_1, c_2, c_3. In other words, if we select code c_1, codes c_2 and c_3 cannot be selected.

Handling of parameters is mostly done in separate systems, where valid values (or sometimes even valid combinations of values) are specified via tables listing the admissible settings.

3. Challenges

Since software has become central in modern cars, the question arises whether existing systems, into which the software configuration is mostly just cobbled in, still meet all the necessary or desired requirements.

Until today, the configuration process is still mostly divided into two steps. First the hardware configuration, then – on top of it – the software configuration. However, it is questionable whether this two-step process is still the best approach for existing and future use-cases.

In this section, we present various challenges that prevail in current hardware-centric configuration systems and might require additional consideration in future hardware-software co-configuration systems.

Hardware Upgrade. Automotive manufactures recently started to offer subscription models for features that are not present at the time of sale, but can be retrofitted – often by a simple software switch – into already delivered vehicles. For this, the manufacturers need the possibility to enable functions for vehicles in the field. Typically, this also requires a check, whether the hardware of the car supports the extended functionality.

It might even be the case that an OEM (Original Equipment Manufacturer) allows a retrofit including an update of hardware components. This can occur as follows: In a revision of an existing car series, the hardware is slightly modified and a new software function becomes available. Now, this software function could potentially also be provided in the older model, if the necessary hardware can be retrofitted. Checking whether such an update is possible (and which parts have to be exchanged) requires access to the exact configuration of the delivered car as well as configuration systems that have knowledge about constraints for historic configurations, possibly going back to several years or even decades.

OTA Update. Over-the-air (OTA) software updates present unique challenges for automotive manufacturers. Firstly, the vehicle's software configuration must be fully defined in both hardware and software terms, ensuring that only compatible software satisfying all dependencies is delivered to the vehicle. Additionally, the delivered software must be correctly configured through variant coding, based on the underlying hardware configuration. Therefore, a combined consideration of hardware and software is highly important

Missing Expert. Up to day, software updates are still performed in repair shops by an expert. Manufacturers profit in this context from the expertise of the working staff. Occurring errors can instantly be analyzed by a qualified person. In the best case, the underlying error can be directly fixed by the repair shop staff. Especially in the case of OTA updates, this expertise is missing. In particular, problem analysis and troubleshooting pose a special challenge, as they all have to be done before delivery of the update or – for residual errors – must be fixable remotely, in the worst case by a downgrade to the previous version.

Certification. The automotive sector is highly standardized and regulated. Automotive manufacturers must guarantee that their products satisfy all kinds of standards from different domains. The regulations classically certify a vehicle to satisfy predefined security and safety standards. With the increasing use of driver assistant systems in the automotive sector, the certification requirements, especially in software, grow. In particular, lawmakers want to know in the future exactly which software is delivered in which car. An example is the UN ECE regulation 156,[1] describing the requirements for a software update management system (SUMS) and the future scope for type approval procedures under consideration of the software.

[1] https://unece.org/transport/documents/2021/03/standards/un-regulation-no-156-software-update-and-software-update

Frequent Updates. Software updates can be much more frequent than hardware revisions. As software is in an increasing manner not only security but also safety relevant, software updates may have the need to be rolled out quickly. However, as software changes can impact other components of the vehicle, an automatic validity and conformity check of the target vehicle configuration is required. In the best case, after a fix in an existing software package, a package manager should compute a new valid configuration, including the fixed software.

Vehicle function distribution. The distribution of functions in a vehicle and their mapping to ECUs is still done mostly manually. However, with the rapidly changing software architectures in vehicles, the distribution is getting more and more complex. To simplify the initial process in development, algorithmic support is an obvious solution. This requires a detailed specification and documentation of dependencies of the software to be distributed. This initial distribution, we call it *static vehicle function distribution*, can also be extended to the dynamic case, in which functions can be (re-)distributed in real-time to corresponding computing nodes. This allows an improved energy usage, as ECUs can be turned off and on if needed – complex algorithms might even be run in the cloud. For both use cases, a complete description of the hardware and software dependencies is required.

Version Constraints. For software configurations, the specific version of a software package and its dependencies are vital information. In a major software release, dependencies might change drastically, reflecting the changed and extended behavior of the software. However, version constraints are mostly documented in a numeric way, e.g. via *Semantic Versioning*.[2] Whether the currently employed Boolean formalization of constraints is still the best way to address the problem is questionable.

Variance over time. If software updates can still be carried out for older vehicles, new functionalities can also be integrated into existing fleets if technically feasible. Determining which vehicle configurations can still be supplied with new software, as well as documenting and verifying this, is a major challenge given the enormous space of possible configurations. Enabling this variance over larger timeframes will require new mechanisms and considerations.

Software Packaging. In classic computer systems, software configuration problems are often solved with the help of package managers. However, automotive software has additional constraints. In addition to the hardware dependency, there are also complex parameterizations (variant coding) and the need for diagnostic options. This raises the question of how to define software components or packages in order to be able to adopt existing concepts.

4. Related Work

How to handle hardware/software configurations in the automotive sector has been an active point of research and discussion for several years now [4, 5]. In a survey of German car manufacturers, Sax et al. [6] claim that new ways of checking the consistency of major, regular software updates is an important aspect for not hindering fast development of new functions in the future.

The configuration problem in the automotive industry and solutions to it were already described in the early 2000s. Sinz [7] describes a rule system based on Boolean logic. Here, not only checking individual configurations is covered, but also ways to determine common properties of all valid configurations. Moreover, the mapping from code sets into concrete physical components is also considered. In a later publication, Sinz [8] also describes the verification of such rule systems in order to detect and minimize errors at an early stage. Astesana et al. [9] on the other hand, describe vehicle configurations by using a CSP framework, with Renault as a case study. More recent publications also deal with the topic of classic configurations. Bischoff et al. [10] describes a graphical editor for visualizing and editing item selection rules.

In addition to the control systems mentioned above, there are other approaches to describing variability in general. One of them is feature modeling. In this, the configuration options (features) are often represented in the form of trees, which represent the relations between the features. The analysis of feature models is often performed with the use of SAT solvers [11]. However, analysis approaches using SMT solvers have also been part of recent research [12].

In the area of classic computer systems, configurations are often found in the area of package management and dependency solving. These are mostly so-called component-based systems such as GNU/Linux distributions (e.g. Debian [3]). These contain metadata for software packages, which the package managers utilize in their search for valid configurations. While most package managers initially used ad-hoc solvers [13], nowadays more efficient algorithms from the SAT or CSP community are usually employed [14]. Pinckney et al. [15] recently proposed a package solver, *PacSolve*, for NPM which uses

[2] https://semver.org

[3] https://www.debian.org

an SMT approach instead of SAT/CSP solvers. As an alternative to SAT and SMT, there are also publications that describe and solve package update problems with Answer Set Programming [16, 17].

The distribution of different functions in the architecture of a vehicle represents an enormous challenge in modern vehicles. Ruhnau et al. [18] take a first step towards mastering this challenge by describing an ontology for function distribution, covering both static and dynamic distribution.

5. Conclusion

In this paper, we have described various challenges that exist in the area of hardware/software configurations in the automotive sector. Many of these challenges arise from the increasing complexity and relevance of software in vehicles. We have listed that research work already exists for some of the topics. For others, there are promising approaches from related problem areas, the suitability of which we will investigate in more detail in future work.

Acknowledgments

The research presented in this paper was done in the context of the SofDCar (19S21002) project, which is founded by the German Federal Ministry for Economic Affairs and Climate Action.

References

[1] C. Fehling, M. Frank, O. Kopp, Digital sustainability and digital diversification: The two key challenges for automotive software development, in: IEEE 18th Intl. Conf. Software Architecture Companion (ICSA-C), 2021, pp. 162–166.

[2] A. Kübler, C. Zengler, W. Küchlin, Model counting in product configuration, in: Proc. of the 1st Intl. Workshop on Logics for Component Configuration, LoCoCo 2010, Edinburgh, UK, 2010, pp. 44–53.

[3] H. Takimizu, T. Fukaya, Y. Ito, N. Sakano, ECU variant coding system, Mitsubishi Motors Technical Review 18 (2006).

[4] O. Media, Effective hardware-software co-design for automotive systems, 2014. URL: https://embeddedcomputing.com/application/automotive/effective-hardware-software-co-design-for-automotive-systems.

[5] O. Burkacky, J. Deichmann, G. Doll, C. Knochenhauer, Effective hardware-software co-design for automotive systems, 2018. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/rethinking-car-software-and-electronics-architecture.

[6] E. Sax, R. Reussner, H. Guissouma, H. Klare, A Survey on the State and Future of Automotive Software Release and Configuration Management, Technical Report 11, Karlsruher Institut für Technologie (KIT), 2017.

[7] C. Sinz, Baubarkeitsprüfung von Kraftfahrzeugen durch automatisches Beweisen, Diplomarbeit, Universität Tübingen, 1997.

[8] C. Sinz, Verifikation regelbasierter Konfigurationssysteme, Ph.D. thesis, Universität Tübingen, Tübingen, Germany, 2003.

[9] J.-M. Astesana, L. Cosserat, H. Fargier, Constraint-based vehicle configuration: A case study, in: 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, 2010, pp. 68–75.

[10] D. Bischoff, W. Küchlin, O. Kopp, Poseidon: A graphical editor for item selection rules within feature combination rule contexts, in: PLM in Transition Times, Springer, Cham, 2023, pp. 3–14.

[11] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (2010) 615–636.

[12] J. Sprey, C. Sundermann, S. Krieter, M. Nieke, J. Mauro, T. Thüm, I. Schaefer, SMT-based variability analyses in FeatureIDE, in: Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems, ACM, 2020.

[13] P. Abate, R. D. Cosmo, R. Treinen, S. Zacchiroli, A modular package manager architecture, Information and Software Technology 55 (2013) 459–474.

[14] P. Abate, R. D. Cosmo, G. Gousios, S. Zacchiroli, Dependency solving is still hard, but we are getting better at it, in: 27th Intl. Conf. Software Analysis, Evolution and Reengineering (SANER), IEEE, 2020.

[15] D. Pinckney, F. Cassano, A. Guha, J. Bell, M. Culpo, T. Gamblin, Flexible and optimal dependency management via Max-SMT, in: 45th Intl. Conf. Software Engineering (ICSE), 2023, pp. 1418–1429.

[16] M. Gebser, R. Kaminski, T. Schaub, aspcud: A linux package configuration tool based on answer set programming, Electronic Proceedings in Theoretical Computer Science 65 (2011) 12–25.

[17] T. Gamblin, M. Culpo, G. Becker, S. Shudler, Using answer set programming for HPC dependency solving, in: Proc. Intl. Conf. on High Performance Computing, Networking, Storage and Analysis, SC '22, IEEE Press, 2022.

[18] J. Ruhnau, M. Sommer, J. Henle, A. Walz, S. Becker, E. Sax, Ontology for vehicle function distribution, in: IEEE Intl. Systems Conf. (SysCon), Vancouver, Canada, 17-20 April 2023, IEEE, 2023, p. 1–6.