# Heterogeneity: A Challenge in Automotive Product Configuration

Daniel Bischoff[1], Carsten Sinz[2,*]

[1]Mercedes-Benz AG, Leibnizstraße 6/1, 71032 Böblingen, Germany
[2]Karlsruhe University of Applied Sciences, Moltkestraße 30, 76133 Karlsruhe, Germany

**Abstract**

Automotive configuration systems have been in productive use for many decades. Although historically mainly dealing with mechanical components, there has been a tremendous shift towards electronic, software- and cloud-based components and companion services over the past years. Systems have been extended accordingly, leading to a divergence in methods used for product description, such that today one has to deal with a variety of heterogeneous techniques and systems. In this paper we describe the current situation in the automotive industry and possible ways towards more uniform formalisms for the future taking, in particular, concepts from programming languages into account.

## 1. Introduction

Cars are complex products assembled from a large number of components, including e.g. motors, tires, seats, and entertainment units. Depending on customer wishes and production sites, the number of variants, in which a particular car model can be produced, often exceeds the number of atoms in the universe.

Over the last years, more and more software features are incorporated into cars leading to an even larger space of configurability, with fast-changing software features and versions, cloud components, and updates.

Configuration data is relevant in many business units and departments of an automotive company, including sales, engineering, production, and after-sales, among others. The systems and formalisms employed are, however, not unified and homogeneous. Instead, different departments use different systems with (often only slightly) differing syntax and semantics. Electronic equipment is handled separately from mechanical components in engineering, the sales department uses simplified models (hiding, e.g. features relevant only to control the production process), factories need additional information about part availability and timing.

Thus, a large number of heterogeneous systems are nowadays present in automotive companies. Differing syntax and semantics in these systems result in complex interfaces and overly complex processes. Thus, working towards a unified product modeling language seems to be a profitable venture.

## 2. Diversity in Automotive Product Configuration

Current systems for automotive variability modeling at many (European) car manufacturers have core characteristics like:

- A large number of (Boolean) configuration options (called *codes* at Mercedes-Benz or *features* in the SPL community), typically around or above a few thousand.
- A large number of constraints (Boolean formulas) between these options, typically in the tens of thousands. Each constraint may include only a few or up to several dozens of codes.

*ConfWS'25: 27th International Workshop on Configuration, Oct 25–26, 2025, Bologna, Italy*
*Corresponding author.
daniel.bischoff@mercedes-benz.com (D. Bischoff); carsten.sinz@h-ka.de (C. Sinz)
0000-0001-7116-9338 (C. Sinz)

- A two-level configuration formalism, consisting of a high-level specification language based on the codes (called *product documentation* at Mercedes-Benz), and a low-level language dealing with parts and their association with the high-level codes. Constraints can only be formulated in the high-level language. Mapping from combinations of high-level codes to parts is accomplished using Boolean formulas over the codes, describing under which conditions a part is included in the car.
- Conceptually, the presence of many different views on the product documentation, e.g., restricted to a particular country a car is shipped to, or to certain product (sub-)lines, e.g. convertibles. Practically, tool support for generating and dealing with views is still limited, though.
- A sprawling and heterogeneous landscape of software systems and tools, often evolving over many years or even decades, leading to inconsistencies and a lack of uniformity. Additional systems are required to configure and maintain parameters of electronic components (e.g., the number of motor steps needed to open a window). Software versions must be managed throughout a vehicle's lifetime, including support for over-the-air updates.

As an example, let's consider the research and development departments of any OEM. They integrate a wide range of data formats, systems, domains, and processes that collectively determine the current "data state" for all products and the production environment.

One system deals with production planning and bill-of-materials (BOM), while another system extends this data with plant-specific views, including, e.g., deployment dates or parts availability.

The types of computations performed on this data includes, among others:

1. *Buildability checks* validate whether complete orders can be built and evaluated.
2. *Completion* of partial vehicle configurations to form valid, buildable vehicles.
3. *Parts requirement analysis*, where "part" can refer to a physical component, a color, or even a software parameter.

The product data is continuously transferred between different departments, extended and aggregated, e.g., for certification or to comply with standards (e.g. WLTP) including measuring emissions, fuel and energy consumption.

## 3. Towards a Unified Language

To integrate different automotive configuration systems we propose a unified language with special features tailored in particular for the automotive domain. Core properties we envisage:

1. *Type system:* The main types are Boolean (reflecting *codes*) and enumeration type (groups of mutually exclusive components). Numerical attributes should also be expressible (e.g. for power consumption) via integer and floating point types.
2. *Constraints:* Complex interactions between Boolean or enumeration variables, as well as numeric properties, should be expressible as rules in a suitable language.
3. *Separation between features and parts level:* There is a separation between the feature space and the parts space with a mapping from (high-level) features to (low-level) parts. Currently, we assume that constraints are solely expressed on the high/feature level.
4. *Modularization:* On the parts level, it should be possible to express nested sub-structures reflecting the cars' assembly process.
5. *Versioning:* Sophisticated versioning should be available (in particular needed for software components), e.g. using semantic versioning.
6. *Timed:* Temporal restrictions should be possible on most entities (rules, parts, features) indicating time intervals in which they are available or enabled.

Our goal is to develop a specification language – or even a programming language – tailored towards product configuration and development, with a particular focus on the automotive industry.

In software development, key challenges such as managing complexity and supporting evolution are central. Programming languages and software management tools (e.g. git) have a long history. We thus aim to explore whether concepts from this domain can be effectively transferred to the context of product development.

## 4. Transferring Concepts from Programming Languages

In this section we want to recapitulate features from programming languages that might also be suitable for describing products and their development.

**Type Systems.** Type systems serve as a means to specify the set of values a variable can take and the structure of this set. So, e.g. *product types* (such as structs or classes) have the effect of an AND-operation (an object needs attribute 1 and attribute 2, etc.), wheres *sum types* (e.g. enumerations) have the effect of an OR-operation. Using both types, in principle, complex Boolean type structures can be built, however, in a cumbersome way.

So extending a type system with sum and product types by special constructs, e.g. extensions or restrictions of enumerations, adding rule-based type restrictions seems appropriate.

**Object-Orientation.** Features and parts can have attributes, often addressed with a dot-syntax, in object-oriented languages. Variables for, e.g., different motors, `M1`, `M2`, modeled as features, could have properties such as displacement, or number of cylinders, which can be described as attributes:

```
M1.displacement = 2.0
M2.nrCylinders = 6
```

An attribute $a$ can be regarded as a functional dependency of an object $o$ (feature or part), which can be addressed as $a(o)$. Typically, there is no ambiguity with which object an attribute is associated.

This is different for rules (a.k.a. constraints), which connect several objects, and where it is often not clear, which is the "main object" the rule should be associated with. Even for simple exclusions such as $\neg(A \wedge B)$ this problem shows up: Should this constraint be associated with $A$ or $B$? For implications $A \implies B$ the constraint is often stored together with the antecedent ($A$). But this has the effect, that either the constraint has to be shown a second time for $B$, or the link of the constraint to $B$ is not visible directly. No simple way seems to be available to solve this association problem. In practice, data engineers develop semi-strict patterns and standards, which, at least for the editing use cases, seem unavoidable. For consumers of the same data, different approaches are available.

Additionally, to alleviate the problem of associating constraints, we propose to group rules based on their *purpose*. There could be groups for legal restrictions in a particular country, for geometrical restrictions based on part geometry, etc.

Such grouping, if nested, enforces an order on which purpose is considered the most important (the outermost group) and which is less important or "smaller" (the innermost group).

Such an enforcement could be avoided by associating rule groups with variables for restrictions, of which multiple could be applied to a group, or *context*, as we will call it now.

```
context Country=USA, Motor=M2 {
  // rules concerned with use of Motor M2 in USA
}

context Country=Austria {
  // rules concerned with Austria
}
```

**Versioning.** Almost every entity in automotive product configuration is subject to changes over time. This includes features, their attributes, constraints between them, parts, and more. Thus versioning should be included in the language, e.g., in the form of semantic versioning, as often used in software development. Moreover, parts can be equipped with further constraints that are added at each plant, e.g., reflecting parts availability.

For complete vehicles, Mercedes and many other OEMs use specific codes to identify the so called model year of vehicles.

**Integration in the Business Process.** A configuration language is always part of a larger business process in practice. We therefore propose an integration approach as follows (cf. Fig. 1): Users generate configuration models using visual tools, specialized for different purposes. These tools translate visual constraints into the unified product configuration language that serves as an integrating formalism. Then, additional tools can be employed to run algorithms on a logical (Boolean) model derived from the configuration language, e.g. to detect errors or inconsistencies in the model, to determine parts requirements, etc.


**Figure 1:** Integrating a unified product configuration language in a business setting.

**Business Perspectives.** Visualization and restricted views are an important aspect to make large knowledge bases comprehensible. To that end, in the Poseidon approach presented in [1], we showed how to use a) partial assignments (for staged configuration in the sense of [2]) on a Boolean formula level, b) projection, and c) configurable variable orderings (ROBDD-likes) to produce concise and customizable data visualizations of such configuration data.

Using Poseidon's techniques, a sales department can, for example, project constraint sets into the subset of variables they care about. Thereby the technical details of *why* some option excludes another might be hidden, only the fact itself remains as a constraint between the sales-level variables. The engineering departments, in turn, can project to a more technical level.

In general, state-of-the-art PLM/PDM systems allow modeling of variability, nowadays often using object-like or predicate-logic notation (e.g. feature models, domain specific configuration languages). These notations typically need to be translated to a suitable logic, e.g. finite-domain logic or propositional logic, for algorithmic processing (e.g. completeness checks, performance in evaluation, etc.) anyway, which is why we propose a common language on the propositional level.

**Modularization and Subsystems.** Modularization in programming languages has several purposes: separation of concerns, improved readability and maintenance, code reuse and simplified collaboration, among others. It is achieved by multiple concepts, varying from one language to another, e.g., by breaking large programs in multiple files, folders or packages; by introducing namespaces, functions or classes; or by adding access modifiers (private variables not visible outside a certain scope).

All these concepts could be transferred to configuration. With textual variability modeling languages, splitting into multiple files and folders seems a simple first approach.

**Some Examples.** We shortly want to give some first ideas, how the syntax of such transferred contexts could look like:

```
group Motor {  // an enumeration type
  M1, M2, M3   // for different motors
}

group SalesTypes {
  S1: M1, T1  // group of three sales
  S2: M2, T1  // types, each with a set of
  S3: M3, T2  // defining component options
}
```

```
group Transmission {
  T1, T2
}

context USA {
  not M1 and not T2
}
```

Following these definitions, an overall high-level constraint, hlc, could be derived, e.g. for a tool used in the sales department. We use pseudo-boolean exactly-one-constraints (EXO) to express necessity and pairwise exclusion for the groups:

$$\begin{aligned}\text{hlc:}\quad & \text{EXO}(\texttt{M1},\texttt{M2},\texttt{M3}) \wedge \text{EXO}(\texttt{T1},\texttt{T2}) \wedge \\ & \text{EXO}(\texttt{S1},\texttt{S2},\texttt{S3}) \wedge (\texttt{S1} \longleftrightarrow \texttt{M1} \wedge \texttt{T1}) \wedge \\ & (\texttt{S2} \longleftrightarrow \texttt{M2} \wedge \texttt{T1}) \ \wedge (\texttt{S3} \longleftrightarrow \texttt{M3} \wedge \texttt{T2}) \wedge \\ & (\texttt{USA} \longrightarrow \neg\texttt{M1} \wedge \neg\texttt{T2})\end{aligned}$$

Now, a business perspective for sales could be calculated by projecting the high-level constraint to the set of sales types (S1, S2, S3) and markets (USA and others). In other words, by existential quantifying out all non-relevant variables (M1, M2, M3, T1, T2) in hlc, we obtain:

$$\begin{aligned}\text{projection}(\text{hlc}, \{\texttt{S1},\texttt{S2},\texttt{S3},\texttt{USA}\}) &= \exists\, \texttt{M1},\texttt{M2},\texttt{M3},\texttt{T1},\texttt{T2} : \text{hlc} \\ &= (\texttt{S1} \wedge \neg\texttt{S2} \wedge \neg\texttt{S3} \wedge \neg\texttt{USA}) \\ &\quad \vee (\neg\texttt{S1} \wedge \texttt{S2} \wedge \neg\texttt{S3}) \\ &\quad \vee (\neg\texttt{S1} \wedge \neg\texttt{S2} \wedge \texttt{S3} \wedge \neg\texttt{USA})\end{aligned}$$

This result shows that both S1 and S3 are not valid in the USA, while S2 has no such restriction. The result is visualized in Fig. 2 using the Poseidon tool.

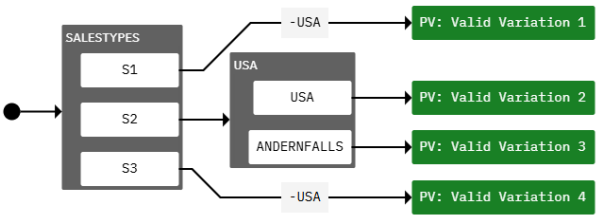


**Figure 2:** A visualization of the validity constraint from a sales perspective derived by projecting the high-level constraint to the set of sales types and markets, sorting those options and compiling them into an (RO)BDD-like structure.

## 5. Related Work

There has been extensive work on formalisms for product configuration, both in textual and graphical form. Due to the vast number of relevant contributions in this area, it is difficult to adequately acknowledge all authors. We thus want to focus on a few. Felfernig et al. provide the foundational material for modeling and composing complex products [3]. Schmid and Eichelberger maintain a list and classification of textual variabiltiy modeling languages, which is publicly available under https://sse.uni-hildesheim.de/en/textual-variability-overview. This list has last been updated in June 2017, but still gives a good reference to the work until then. On this website they also cover, e.g.,

scalability support (D.4). Another nice overview is given by Beek *et al.* [4]. The Universal Variability Language (UVL) is a community effort towards a unified language for variability models [5, 6]. These languages are intended to be "general-purpose", in that they do not focus on a particular industry.

There have also been proposals tailored towards the automotive industry, e.g. by Zellmer *et al.* [7], Visser *et al.* [8] or Jost and Sinz [9].

## 6. Conclusion

This short paper is intended to reflect the current state of product configuration in automotive industry practice, which is characterized by many non-uniform specification mechanisms and accompanying systems. This heterogeneity leads to challenges in maintaining both systems and product data, as well as for integrating new requirements – particularly those arising from the increasing shift toward software-based components.

We assembled a set of properties that we consider important in a formalism for integrated, large-scale automotive product configuration; however, no project has yet been launched to realize the approach, and its scalability still needs to be assessed in practice.

Moreover, while the extent to which our approach will be adopted by industrial practitioners remains to be demonstrated, we regard the prospects as promising. With the integration approach outlined above (see Fig. 1) and the support of intuitive visual tools, we anticipate that acceptance in industry can be greatly strengthened.

**Acknowledgments.** We would like to thank Christian Seiler and Klaus Anwender from Mercedes-Benz AG for fruitful discussions on type-based product line engineering (TPLE) for Mercedes' cars.

**Declaration on Generative AI.** During the preparation of this work, the authors used a Mercedes-internal GenAI tool (based on GPT-4o) and Thesify for grammar, spelling checks, and literature proposals. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

## References

[1] D. Bischoff, W. Küchlin, O. Kopp, Poseidon: A graphical editor for item selection rules within feature combination rule contexts, in: IFIP International Conference on Product Lifecycle Management, Springer, 2022, pp. 3–14.

[2] K. Czarnecki, S. Helsen, U. W. Eisenecker, Staged configuration using feature models, in: R. L. Nord (Ed.), Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings, volume 3154 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 266–283. URL: https://doi.org/10.1007/978-3-540-28630-1_17. doi:10.1007/978-3-540-28630-1_17.

[3] A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen, Knowledge-based configuration: From research to business cases, Newnes, 2014.

[4] M. H. t. Beek, K. Schmid, H. Eichelberger, Textual variability modeling languages: An overview and considerations, in: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B, SPLC '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 151–157. doi:10.1145/3307630.3342398.

[5] C. Sundermann, K. Feichtinger, D. Engelhardt, R. Rabiser, T. Thüm, Yet another textual variability language? a community effort towards a unified language, in: Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A, SPLC '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 136–147. URL: https://doi.org/10.1145/3461001.3471145. doi:10.1145/3461001.3471145.

[6] C. Sundermann, S. Vill, T. Thüm, K. Feichtinger, P. Agarwal, R. Rabiser, J. A. Galindo, D. Benavides, UVLParser: Extending UVL with language levels and conversion strategies, in: Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B, SPLC '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 39–42. doi:10.1145/3579028.3609013.

[7] P. Zellmer, L. Holsten, T. Leich, J. Krüger, Product-structuring concepts for automotive platforms: A systematic mapping study, in: Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A, SPLC '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 170–181. doi:10.1145/3579027.3608988.

[8] R. Visser, A. Basson, K. Kruger, An architecture for the integration of product and production digital twins in the automotive industry, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 431–441. doi:10.1145/3652620.3688257.

[9] F. Jost, C. Sinz, Handling automotive hardware/software co-configurations with integer difference logic, in: Proceedings of the 18th International Working Conference on Variability Modelling of Software-Intensive Systems, VaMoS '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 103–111. doi:10.1145/3634713.3634728.

[10] P. Ochs, T. Pett, I. Schaefer, Consistency is key: Can your product line realise what it models?, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 690–699. doi:10.1145/3652620.3687812.

[11] M. Eggert, K. Günther, J. Maletschek, A. Maxiniuc, A. Mann-Wahrenberg, In three steps to software product lines: a practical example from the automotive industry, in: Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A, SPLC '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 170–177. doi:10.1145/3546932.3547003.