Variability Exchange Language– A Generic Exchange Format for Variability Data

Michael Schulze, Robert Hellebrand
pure-systems GmbH
Agnetenstraße 14
D-39106 Magdeburg
{michael.schulze,robert.hellebrand}@pure-systems.com

Abstract:
The purpose of the *Variability Exchange Language* is to support the information exchange among variant management tools on the one hand and systems development tools on the other hand. The essential tasks of a variant management tool are to represent and analyze the variability of a system abstractly and to define system configurations by selecting the desired system features. A system development tool captures information of a specific kind, such as requirements, architecture, component design, or tests. In order to support the development of variable systems, development tools either have to offer the capability to express and deal with variability directly, or an additional piece of software like an add-on must be provided that adds this capability to the development tool.

To interconnect variant management with systems development, the information exchange among the corresponding tools must be established. A variant management tool must be able to read or extract the variability from a development tool and to pass a configuration, i.e. a set of selected system features, to the development tool. Up to now, the interfaces that support this information exchange are built for each development tool anew. With a standardized *Variability Exchange Language*, a common interface can be defined that is implemented by the development tools and used by the variant management tools. The integration of variant management tools with systems development tools via this interface enables a continuous development process for variable systems and supports a flexible usage of tools for this process.

1 Introduction and Motivation

Variant management is an activity that accompanies the whole system development process. Therefore, it is orthogonal to the other development tasks. Like safety, security, and other system properties, variability cannot be built into a system at the end of the process. Rather, the desired variability has to be determined, analyzed, designed, implemented, and tested continuously, starting at the very beginning of the process, through to the final delivery of the system or the system variant respectively. That means that within each development stage – requirements analysis, design, implementation, test, documentation, etc. – variability is an aspect that has to be considered.

It is an accepted best practice to define an explicit abstract variability model ([BRN+13]) on a system under development to support variant management continuously throughout the process. This abstraction, often specified using feature models[KCH+90], contains the bare information on the variability of the system. That means it describes which variability exist, but it does not describe how the variability is realized in the artifacts. Locations inside an artifact, that are influenced through variability, are denoted as variation points. As example, consider a requirements document specifying a variable system with an optional requirement representing the variation point. In this case two system variants can be formed by either having the requirement or not.

As a side note, we do not specify here how variation points are represented or realized in the artifacts. Some artifact formats support the definition of variation points e.g AUTOSAR [AUT09, SWB12, SK13], in other cases appropriate means have to be added. This obviously also has an impact on the tools that are used to create and manage the artifacts. In some cases they are capable to express variation points. In other cases an add-on has to be built in order to incorporate variation points [PS14].

The abstract variability information has to be connected with the system development artifacts to define how feature selections (system configuration) determine the resolution of the variation points within these artifacts, i.e. the selection of a variation for each variation point. As soon as these connections are established, a feature selection can be carried over to a configuration of the variation points of the concerned artifact. The technical realization – more precise the exchanged data structures, types, and semantics to enable such interactions between a variant management tool and development tools – is addressed by the *Variability Exchange Language*.

To the best knowledge of the authors, at present there is no standard that would define how variation points are consistently expressed independent of the actual artifacts. That means that a tool supplier who builds a variant management tool has to implement an individual interface to each other tool that is used in a development process to create the corresponding artifacts. The purpose of the *Variability Exchange Language* is to support the standardization of these interfaces by a common exchange format that defines, which information is exchanged between a variant management tool and a tool that is used to manage a specific kind of artifacts in a development process. As mentioned above, such a tool may either be a tool that already supports the definition of variation points for the concerned artifact type, or it may be an add-on that provides this capability to a base tool.

In fact, the *Variability Exchange Language* defines a requirement on tools or tool add-ons that intend to support variant management. Such a tool has to be able to extract the data that is defined in the *Variability Exchange Language*, from the artifact that it manages and to incorporate the data that is sent from the variant management tool into this artifact. Beyond the exchange format, i.e. the contents of the information that is exchanged, also some basic operations are defined (see Section 2 and specification [GRHS15]). They define in which direction the variability information is intended to flow.

A use case for the *Variability Exchange Language* is shown in Figure 1. For instance, an artifact created with tool A contains variation points. First, the development tool collects the data, essentially the variation points formatted according to *Variability Exchange Lan-*

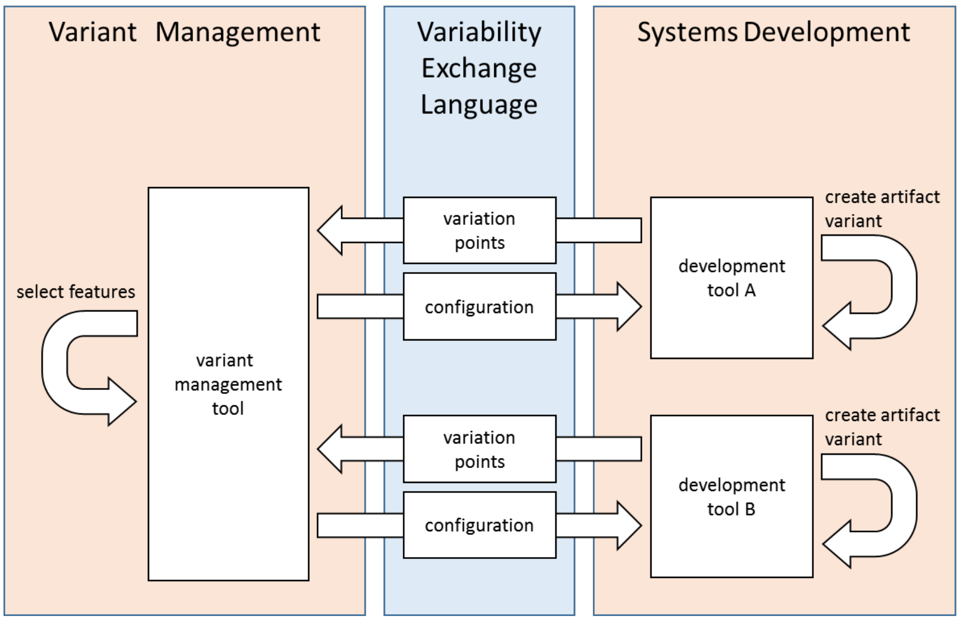

Figure 1: Use case for the *Variability Exchange Language*

guage, and passes this data to the variant management tool that builds a variability model based on the data. This model, in conjunction with the feature model, is used to define a system configuration, meaning to determine which variation for a variation point has to be bound. The corresponding data, i.e. the configuration, again formatted according to the *Variability Exchange Language*, is then passed back to the development tool or add-on that processes this data to derives an artifact variant that corresponds to the system variant defined in the variant management tool.

Applying this scenario to all development tools and artifacts yields a consistent set of development artifacts for any system variant automatically. The variation points that correspond to customer relevant system features should coincide in all artifacts, i.e. they always induce the same variability model in the variant management tool. In addition to that, there may also be internal variation points, for instance implementation variants that do not alter the visible properties of the system, but are relevant for the system construction process. These variation points give rise to a staged variability model, in which customer features are separated from internal features.

The rest of the paper is structured as follows: In Section 2 we give an overview of the concepts and possibilities of the *Variability Exchange Language*. An example application of the *Variability Exchange Language* is described in Section 3. Related work is regarded in Section 4. Finally, the paper is concluded in Section 5.

2 Overview of the *Variability Exchange Language*

The core of the *Variability Exchange Language* is given by the definition of variation points and their variations – by the classes VARIATIONPOINT and VARIATION (see Figure 2). In the following, we immediately use the class names from the meta-model presented in Figure 2 to discuss the corresponding concepts, such as VARIATIONPOINT and VARIATION. A detailed specification of all classes is provided in [GRHS15] and the release of the specification is scheduled in 2015 within the SPES XT project.

VariationPoints and Variations: The *Variability Exchange Language* distinguishes between two kinds of variation points (see Figure 2). This distinction allows clearly to describe whether the variability belongs to structural or parametric parts.

1. STRUCTURALVARIATIONPOINT – variation points where the structure of a model changes during the binding process. A STRUCTURALVARIATIONPOINT defines which elements are contained in an artifact. There are two kinds of structural variation points:
   (a) OPTIONALSTRUCTURALVARIATIONPOINT – variation points that can themselves be selected or deselected.
   (b) XORSTRUCTURALVARIATIONPOINT – variation points that represent sets of alternatives from which exactly one can be selected.
2. PARAMETERVARIATIONPOINT – variation points which select a numerical value for a parameter during the binding process. They do not change the structure of an artifact. There are two kinds of parameter variation points:
   (a) CALCULATEDPARAMETERVARIATIONPOINT – variation points where the parameter value is calculated by an expression.
   (b) XORPARAMETERVARIATIONPOINT – variation points where the parameter value is selected from a list of values.

Variation points are associated with at least one variation and for each variation point subtype a corresponding variation subtype exists. Variations specify the different manifestations of their respective variation points. To define an artifact variant, the contained variation points are bound by selecting for each variation point one of its variations or even also zero in the case of OPTIONALSTRUCTURALVARIATIONPOINT.

A variation of a variation point comes with a *condition* or an *expression* that determines when or how it is selected (STRUCTURALVARIATIONPOINT) or its value is determined (PARAMETERVARIATIONPOINT). In this case, a *condition* is simply an EXPRESSION which returns a Boolean value. An EXPRESSION can either be a simple Boolean expression or a complex expression which is as powerful as an expression in a typical programming language.

In order to enable referring to artifact's elements, both, variations and variation points, may contain an optional member *correspondingVariableArtifactElement*. Elements referenced

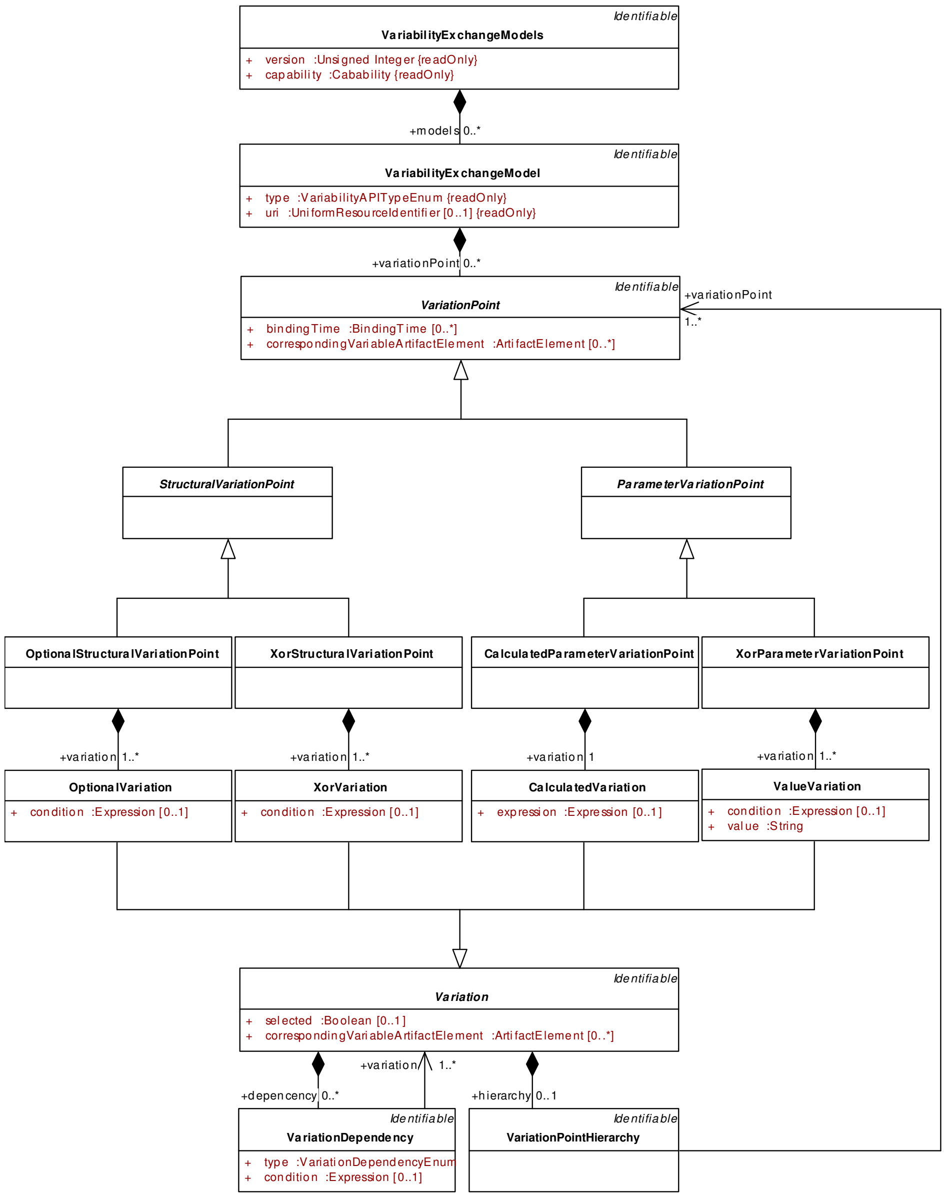
Identifiable
VariabilityExchangeModels
+ version :Unsigned Integer {readOnly}
+ capability :Cabability {readOnly}
+models 0..*
Identifiable
VariabilityExchangeModel
+ type :VariabilityAPITypeEnum {readOnly}
+ uri :UniformResourceIdentifier [0..1] {readOnly}
+variationPoint 0..*
Identifiable
VariationPoint
+ bindingTime :BindingTime [0..*]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
+variationPoint
1..*
StructuralVariationPoint
ParameterVariationPoint
OptionalStructuralVariationPoint
XorStructuralVariationPoint
CalculatedParameterVariationPoint
XorParameterVariationPoint
+variation 1..*
+variation 1..*
+variation 1
+variation 1..*
OptionalVariation
+ condition :Expression [0..1]
XorVariation
+ condition :Expression [0..1]
CalculatedVariation
+ expression :Expression [0..1]
ValueVariation
+ condition :Expression [0..1]
+ value :String
Identifiable
Variation
+ selected :Boolean [0..1]
+ correspondingVariableArtifactElement :ArtifactElement [0..*]
+variation 1..*
+depencency 0..*
+hierarchy 0..1
Identifiable
VariationDependency
+ type :VariationDependencyEnum
+ condition :Expression [0..1]
Identifiable
VariationPointHierarchy

Figure 2: An Overview of the *Variability Exchange Language* [GRHS15]

by this member could be for instance an optional Simulink block or a number of C-source code lines framed by #ifdef and #endif. To support also nesting of e.g. #ifdefs or Simulink subsystems, variation points can establish *hierarchy* (VARIATIONPOINTHIERARCHY). Furthermore, variations of different variation points are depending on or conflicting with each other sometimes, hence variations allow to define dependencies via the optional *dependency* member (VARIATIONDEPENDENCY).

Variation Point Descriptions versus Variation Point Configurations: A VARIABILITYEXCHANGEMODEL , as defined in Figure 2 , can actually serve two different purposes:

- A variation point description lists all variation points and all their variations; that is it describes the complete product line of the respective artifact.
- A variation point configuration is very similar to the variation point description, however, selects one (or zero for OPTIONALSTRUCTURALVARIATIONPOINT) variation for each variation point. The attribute *selected* of VARIATION is used for that purpose. Any such selection must be consistent with the *expression* or *condition* attribute of a variation.

Both variation point description and variation point configuration use the same structure; the attribute *type* of VARIABILITYEXCHANGEMODEL specifies how the model needs to be interpreted.

Binding: The *Variability Exchange Language* does not make any assumptions about how the associated artifact is bound. We do however provide a way to attach Conditions or Expressions to Variations:

- In a STRUCTURALVARIATIONPOINT, a variation comes with a *condition* that states whether the associated artifact elements are part of a bound artifact.
- In a PARAMETERVARIATIONPOINT, a variation determines a value for the associated artifact element. This can be done in two way either by computing the value (CALCULATEDVARIATION) or by selecting a value from a predefined set of values (VALUEVARIATION).

In a variation point description (see previous paragraph), the result of the evaluation of a *condition* or *expression* in a variation must be compatible with the attribute *selected* of a variation. For example, if a variation gets selected, then its condition evaluates to *true*.

Common Concepts: Most classes in the *Variability Exchange Language* are based on the class IDENTIFIABLE, which provides them with a name and a unique identifier. Beside that, IDENTIFABLE also provide a way to attach application-specific data (SPECIALDATA) to elements in the *Variability Exchange Language*.

```
1  #if A
2      /* code active if A is defined          */
3  #if B
4      /* code active if A and B is defined    */
5  #endif
6      /* code active if A is defined          */
7  #else
8      /* code active if A is not defined      */
9  #endif
```

Figure 3: Source code of the file `example.c` where the C-preprocessor is used to realize variability

Exchange Format: To exchange the data expressed in a VARIABILITYEXCHANGE-MODEL between development tools and the variant management tool and vice versa, a serialization is needed. Hence, the *Variability Exchange Language* defines an XML schema [GRHS15].

3 Example Application

To demonstrate the applicability of the *Variability Exchange Language* for the exchange of variability information, we show as an example a simple source code section (see Figure 3), in which the C-preprocessor (cpp) is employed to realize variability, and the extract of that variability defined according to *Variability Exchange Language* (see Figure 4). We could have used further artifact types like requirements, UML models, tests, etc. but for the sake of simplicity and understandability we opted for C-source code using the cpp.

To note, the cpp is a stand-alone tool for text processing, which, although initially invented for C, is not limited to a specific language and can be used for arbitrary text and source code transformations [Fav96], leading e.g. to conditional code compilation. The cpp tool works on the basis of directives (a.k.a. macros) that control syntactic program transformations. The directives supported by the cpp tool can be divided into four classes: file inclusion, macro definition, macro substitution, and conditional inclusion.

In our example, we only use the macros A and B and conditional inclusion mechanisms. The source code comments in Figure 3 explain how the cpp will transform the code depending on the definition of the macros. From an abstract point of view, the code contains two variation points and three variations, which is reflected according to the *Variability Exchange Language* in the exchange format definition in Figure 4. The first variation point spans the source lines 1-9 and contains two alternative variations. The variation point's corresponding elements of the artifact – in this case exactly the source lines – are represented in the exchange format as well. Regarding the variations, the first one (lines 2-6) will be selected if macro A is defined. Otherwise the second variation (line 8) gets selected. Assuming that the macro names are identically with features or at least there exists a mapping from a feature to a macro name, then the *condition* in the eighth line in Figure 4 is the equivalent to the first source code line.

```
 1  ...
 2  <variability-exchange-model type="variationpoint-description" id="
        model" uri="file///c:/example.c">
 3    <xor-structural-variationpoint id="vp1">
 4      <corresponding-variable-artifact-element type="src-lines">
 5          <src-lines>1-9</src-lines>
 6      </corresponding-variable-artifact-element>
 7      <variation id="vp1v1" >
 8        <condition type="single-feature-condition">A</condition>
 9        <corresponding-variable-artifact-element type="src-lines">
10          <src-lines>2-6</src-lines>
11        </corresponding-variable-artifact-element>
12        <hierarchy id="vp2h1">
13          <variationpoint ref="vp2"/>
14        </hierarchy>
15      </variation>
16      <variation id="vp1v2">
17        <corresponding-variable-artifact-element type="src-lines">
18          <src-lines>8</src-lines>
19        </corresponding-variable-artifact-element>
20      </variation>
21    </xor-structural-variationpoint>
22    <optional-structural-variationpoint id="vp2">
23      <corresponding-variable-artifact-element type="src-lines">
24          <src-lines>3-5</src-lines>
25      </corresponding-variable-artifact-element>
26      <variation id="vp1v1" >
27        <condition type="single-feature-condition">B</condition>
28        <corresponding-variable-artifact-element type="src-lines">
29          <src-lines>4</src-lines>
30        </corresponding-variable-artifact-element>
31     </variation>
32    </optional-structural-variationpoint>
33  </variability-exchange-model>
34  ...
```

Figure 4: The *Variability Exchange Language* representation of `example.c` from Figure 3

The second variation point (lines 3-5) is nested within the first variation of the first variation point, constituting a variation point hierarchy. Within the corresponding exchange format, the variation points are not nested but the nesting information is covered by the definition in the lines 15-17 in Figure 4. There, the nested variation point is referenced by its *id*, resulting in a tree-like structure at the end.

4 Related Work

Related work in the area of languages and exchange formats in general is manifold, but usually is more centered around the exchange of application data like geographical information (GPS Exchange Format – GPX) or multimedia data (Broadcast Metadata Exchange Format – BMF) to name just two. However, if the context is more on data needed for developing systems, the picture looks different. Focusing further on the exchange of development data supporting variability information, the number of existing related work is rather low.

Some standards like AUTOSAR [AUT09], EAST-ADL [eas], or IP-XACT [iee10] define exchange formats and to different extents it is possible to cope with variability. For example, IP-XACT has a notion of a *variant*, enabling the specification of those elements that are belonging to a variant. What is not supported is to describe which elements are variable at all. In contrast to IP-XACT, EAST-ADL has the notion of variation points and thus provides the functionality to indicate model elements as variable but on the other side EAST-ADL has no notion of a variant. AUTOSAR supports both concepts and thus is the most complete solution with respect to variability. The downside even of the AUTOSAR solution is that the provided possibilities are not generic enough to be used in other contexts as for which they were designed for.

A promising language for the generic description of variability is the Common Variability Languages (CVL) [Øy12] , since CVL provides all needed concepts to cope with variability in a generic, and where possible, artifact independent way. The disadvantage of the language specification is the lack of a serialized data representation, which may be usable for the exchange between tools. In contrast to CVL, which describes how to realize variability description inside a tool, VEL specifies a communication interface and a data exchange format. The interpretation and use of the exchanged data remains the responsibility of the tool providers.

5 Conclusion and Future Work

Tools for variant management frequently interact with artifacts such as model based specifications, program code, or requirements documents. This is often a two-way communication: variant management tools import variability information from an artifact, and in return export variant configurations. For example, they need to gather information about the variation points that are contained in the artifact, need to know which variants are already defined, and then modify existing or define new variants ("this variation point stays, that one goes away") or even define new variation points ("artifact elements a,b and c are alternatives"),

There is currently no standardized exchange format available for such scenarios. Hence, a variant management tool needs to implement a separate one for each new artifact or development tool. Worse, each variant management tool needs to do this separately. With m variant management tools and n artifacts, this may require the implementation of up to $m * n$ different variability data representation and interfaces.

In this document, we presented a generic *Variability Exchange Language* that allows variant management tools to communicate with artifacts through a standardized *Variability Exchange Language*. If realized across both, variant management tool and development tools, this may reduce the number of required implementations significantly. A generic realization typically also lowers the barrier for adding new development tools, and fosters the introduction of new tools for variant management.

6 Acknowledgements

Michael Himsolt (Daimler AG), Martin Große-Rhode (FOKUS), Stefan Mann (FOKUS), Ina Schäfer (TU Braunschweig), Sandro Schulze (TU Braunschweig), Christian Gebhart (Berner&Mattner), Monika von der Werth (Cassidian) and Mihail Constantinescu (Cassidian) have contributed to the *Variability Exchange Language* specification.

This research has been supported by the Federal Ministry of Education and Research in project SPES XT (funding id: 01IS12005). The responsibility for the contents rests with the authors.

References

[AUT09] AUTOSAR. AUTOSAR: Generic Structure Template. Specification, 2009.

[BRN+13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, page 1, Italy, 2013. ACM Press New York, NY, USA.

[eas] EAST-ADL Association.

[Fav96] Jean-Marie Favre. Preprocessors from an abstract point of view. In , *International Conference on Software Maintenance 1996, Proceedings*, pages 329–338, November 1996.

[GRHS15] Martin Große-Rhode, Michael Himsolt, and Michael Schulze. The Variability Exchange Language. Specification, 2015.

[iee10] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows. *IEEE Std 1685-2009*, pages 1–374, February 2010.

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, Wiliam E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University, Software Engineering Institute, Pittsburg, PA, USA, November 1990. (CMU/SEI-90-TR-21).

[PS14] Maria Papendieck and Michael Schulze. Concepts for Consistent Variant-Management Tool Integrations. 2014.

[SK13] Michael Schulze and Stefan Kuntz. AUTOSAR and Variability. *ATZextra worldwide*, 18(9):108–110, 2013.

[SWB12] Michael Schulze, Jens Weiland, and Danilo Beuche. Automotive model-driven development and the challenge of variability. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 207–214, New York, NY, USA, 2012. ACM.

[Øy12] Hagen et al. Øystein. Common Variability Language (CVL). Specification, OMG, 2012.