BVR – Better Variability Results

Øystein Haugen[1] and Ommund Øgård[2]

[1] SINTEF, PO Box 124 Blindern, NO-0314 Oslo, Norway
oystein.haugen@sintef.no
[2] Autronica Fire & Security, P.O. Box 5620, NO-7483 Trondheim, Norway
Ommund.Ogaard@autronicafire.no

Abstract. We present BVR (Base Variability Resolution models), a language developed to fulfill the industrial needs in the safety domain for variability modeling. We show how the industrial needs are in fact quite general and that general mechanisms can be used to satisfy them. BVR is built on the OMG Revised Submission of CVL (Common Variability Language), but is simplified and enhanced relative to that language.

Keywords: Variability modeling · Typing · BVR · CVL

1 Introduction

BVR (Base Variability Resolution models) is a language built on the Common Variability Language (CVL) [1-3] technology, but enhanced due to needs of the industrial partners of the VARIES project[1], in particular Autronica. BVR is built on CVL, but CVL is not a subset of BVR. In BVR, we have removed some of the mechanisms of CVL that we are not using in our industrial demo cases that apply BVR. We have also made improvements to what CVL had originally.

Our motivation has mainly been the Fire Detection demo case at Autronica, but we have also been inspired by the needs of the other industrial partners of VARIES through their expressed requirements to a variability language.

This paper contains a quick presentation of the Common Variability Language in Chapter 2. In Chapter 3, we relate our work to its motivation in the Autronica fire alarm systems, but argue that we need a more compact and pedagogical example and our car case is presented in Chapter 4. Then we walk through our new BVR concepts in Chapter 5, discuss the suggested improvements in Chapter 6, and conclude in Chapter 7.

2 CVL – the Common Variability Language

The Common Variability Language is the language that is now a Revised Submission in the Object Management Group (OMG) [3] defining variability modeling and the

[1] http://www.varies.eu

means to generate product models. CVL is in the tradition of modeling variability as an orthogonal, separate model such as Orthogonal Variability Model (OVM) [4] and the MoSiS CVL [1], which formed one of the starting points of the OMG CVL. The principles of separate variability model and how to generate product models are depicted in Fig. 1

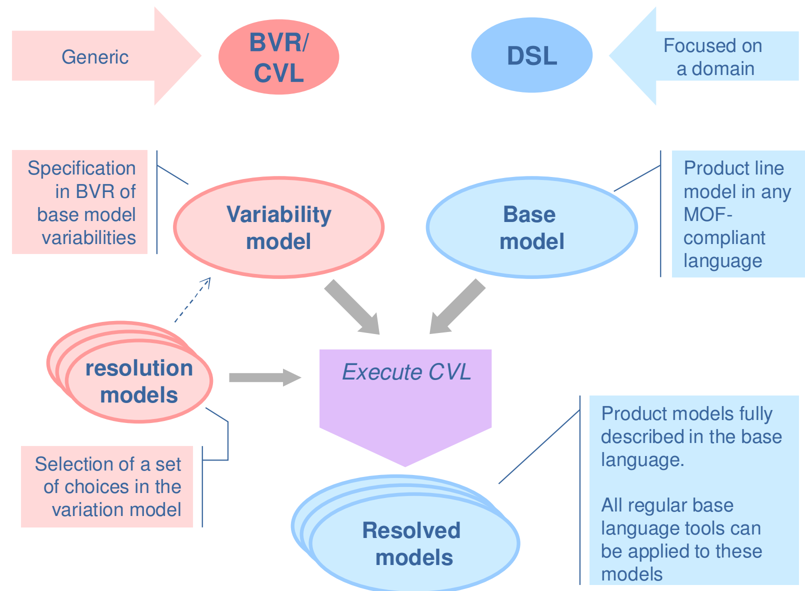

Fig. 1. CVL principles

The CVL architecture is described in Fig. 2. It consists of different inter-related models. The variability abstraction consists of a VSpec model supplemented with constraints, and a corresponding resolution model defining the product selections.

The variability realization contains the variation points representing the mapping between the variability abstraction and the base model such that the selected products can be automatically generated. The configurable units define a layer intended for module structuring and exchange. In this paper we have not gone into that layer.

The VSpec model is an evolution of the Feature-Oriented Domain Analysis (FODA) [5] feature models, but the main purpose of CVL has been to provide a complete definition such that product models can be generated automatically from the VSpec model, the resolution model and the realization model.

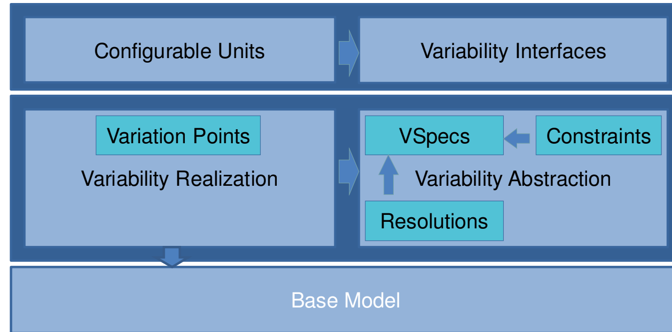

Fig. 2. CVL architecture

BVR (named from Base, Variability, Resolution models) is an evolution from CVL where some constructs have been removed for improved simplicity and some new constructs have been added for better and more suited expressiveness. The new constructs are those presented in this paper.

3 The Autronica Fire Detection Case

The main motivator has been the Autronica Fire Detection Case. Autronica Fire & Security[2] is a company based in Trondheim that delivers fire security systems to a wide range of high-end locations such as oil rigs and cruise ships. Their turnover is around 100 MEUR a year.

The Autronica demo case is described schematically in Fig. 3.

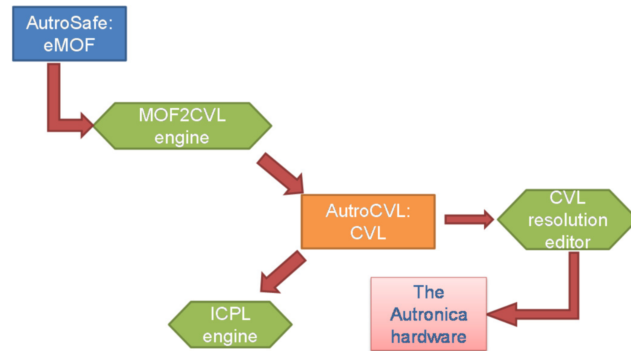

Fig. 3. The Autronica demo case

The purpose of the demo case was to explore the ways in which the Autronica specific model "AutroSafe" could be applied for two different purposes. Firstly, after

[2] http://www.autronicafire.com

transforming the MOF metamodel into CVL the CVL tools could be used to define AutroSafe configurations. Secondly, and possibly more interestingly, from the CVL description it would be possible to apply analysis tools to AutroSafe which were made generally for CVL. In particular the ICPL tool [6-8] could be used to find an optimal set of configurations to test AutroSafe.

For our purpose in this paper, the Autronica use-case provided the real background for understanding what kinds of product line they have to manage. In performing our use case at Autronica, we explored the transition from the AutroSafe model to a CVL model. The AutroSafe model was a UML model that can be understood as a reference model or a conceptual model of how the fire detection system concepts are associated [9]. We realized that this conceptual model could be considered a metamodel, which could be used to generate language specific editors that would be limited to describing correct fire detection systems. Furthermore, we realized that the conceptual model could be used as base for a transformation leading to a variability model. We explored this route by manually transforming the AutroSafe metamodel through transformation patterns that we invented through the work. At the same time Autronica explored defining variability models for parts of the domain directly, also for the purpose of using the variability model to generate useful test configurations.

4 The Example Case – The Car Configurator

Since the Autronica case is rather large and requires special domain knowledge we will illustrate our points with an example case in a domain that most people can relate to, namely to configure the features of a car.

Our example case is that of configuring a car. In fact our starting point for making the variability model was the online configurator for Skoda Yeti in Norway[3], but we have made some adaptations to suit our purpose as example.

Our car product line consists of diesel cars that can have either manual or automatic shift. The cars with automatic shift would only be with all wheel drive (AWD) and they would need the 140 hp engine. On the other hand the cars with manual shift had a choice between all wheel drive and front drive. The front wheel drive cars were only delivered with the weaker 110 hp engine, while the all wheel drive cars had a choice between the weak (110 hp) or the strong (140 hp) engine.

Following closely the natural language description given above, we reach the CVL model shown in Fig. 4.

[3] http://cc-cloud.skoda-auto.com/nor/nor/nb-no/

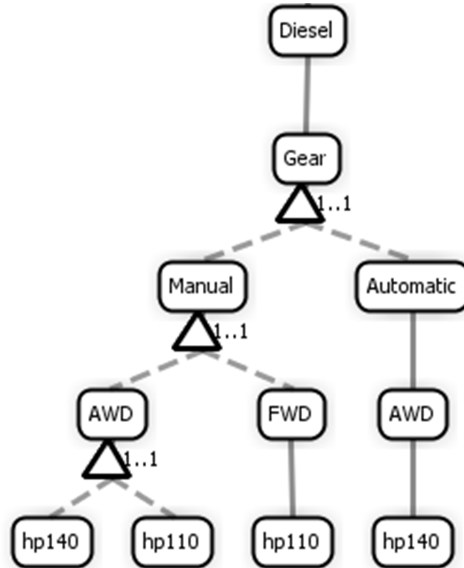

Fig. 4. The example diesel car in CVL

Readers unfamiliar with CVL should appreciate that solid lines indicate that the child feature (or VSpecs as we call them in BVR/CVL) is mandatory when the parent is present in a resolution. Dashed lines on the other hand indicate optionality. A small triangle with associated numbers depicts group multiplicity giving the range of how many child VSpecs must and can be chosen. Thus when *AWD* has children *hp140* and *hp110* associated with a group multiplicity of 1..1, this means that if *AWD* is chosen, at least 1 and at most 1 out of *hp140* and *hp110* must be selected.

5 The BVR Enhancements

In this chapter, we will walk through the enhancements that we have made to accommodate for general needs inspired by and motivated by industrial cases.

5.1 Targets – the Power of the Variability Model Tree Structure

Our CVL diagram in Fig. 4 is not difficult to understand even without the natural language explanation preceding it given some very rudimentary introduction to CVL diagrams (or feature models for that matter). We see that the restrictions are transparently described through the tree structure and our decisions are most easily done by traversing the tree from the top.

It is also very obvious that the diesel car has only one engine, and that it has only one gear shift and one kind of transmission. Therefore everybody understands that even though there are two elements named "hp140" they refer to the same target,

namely the (potential) strong engine. In the same way "AWD" appears twice in the diagram, but again they both refer to the same target. It turns out that CVL and other similar notations do not clearly define this. In fact CVL defines that the two choices named "hp140" are two distinct choices with no obvious relationship at all.

When does this become significant? Does it matter whether the two choices refer to the same target? It turns out that it does both for conceptual reasons and technical ones.

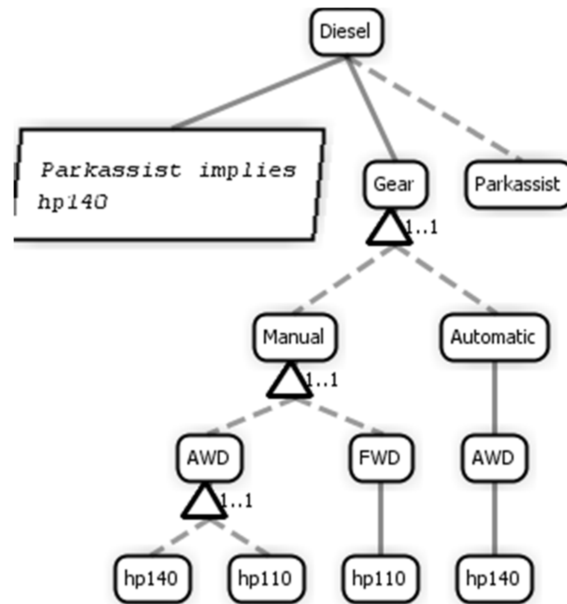

Fig. 5. Adding a Parking assistant

In Fig. 5, we have added an optional parking assistant to the car. However, to be allowed a parking assistant, you need to buy the strong engine. This is intuitive and easily understood, but formally this means that any of the occurrences of "hp140" should satisfy the constraint. Thus, constraints talk about the targets and not the choices.

5.2 Beyond One Tree

We see that the tree structure of variability models convey in a very transparent way the restrictions of the decisions to be made. However, trees are sometimes not enough. In our Autronica experiment, we wanted to reflect in the CVL model the structure of variability in a way that would abstract the actual configurations of fire detection systems in airports and cruise ships. In this way our variability model

became close to the structures of the base model. Our car example model has the opposite focus as it highlights the restrictions of interrelated decisions.

In variability models that are close to the base model, one can expect that tree structures are insufficient to describe the necessary relationships and in the Autronica case the physical layout of detectors and alarms was overlaid by an equally important structure of logical relationships and groups. To represent the alternative, overlaid structures, we need ways to refer between variability elements and our obvious suggestion is to introduce references (or pointers as they are also called).

References can also serve as traces and indicate significant places in other parts of the model.

In our Autronica experiment we had to encode references since references were not available as a concept in CVL. To encode references, we used integers to indicate identifiers and corresponding pointers. This required a lot of manual bookkeeping that turned out to be virtually impossible to perform and even more impossible to read.

In BVR, we want to reflect the physical structure that is represented in the conceptual model as composition through the main hierarchical VSpec tree. The logical structure that is modeled by associations in the conceptual UML model would be represented by variability references in BVR.

5.3 From Proper Trees to Properties

Judging from the tool requirements elicited from the VARIES partners, they wanted a lot of different information stored in the variability (and resolution) models. Some of the information would be intended for their own proprietary analysis tools, and sometimes they wanted to associate temporary data in the model.

When working with the Autronica case and experiencing the difficulties with encoding the needed references we ourselves found that we wished that we had a way to explain the coding in a natural language sentence. Thus we felt the very common need for having comments.

5.4 Reuse and Type – the First Needs for Abstraction

Fire alarming is not trivial. Autronica delivers systems with thousands of detectors and multiple zones with or without redundancy to cruise ships and oil rigs where running away from the fire location altogether is not the obvious best option since the fire location is not easily vacated. In such complicated systems it was not a big surprise that recurring patterns would be found.

Without going into domain-specific details, an AutroSafe system will contain IO modules. Such IO modules come in many different forms and they represent a whole product line in its own right; this actually applies for most of the parts a fire alarm system is composed of, e.g., smoke detectors, gas detectors, panels, etc. Some IO modules may be external units and such external units may appear in several different contexts. As can be guessed, external units have a very substantial variability model and it grows as new detectors come on the market.

In our experiment, we encoded these recurring patterns also by integers as we did with references with the same plethora of integers and need for bookkeeping as a result. It was clear that concepts for recurring patterns would be useful in the language. We investigate introducing a type concept combined with occurrences referring the types.

Our example car product line has no complicated subproduct line, but we have already pointed out that *AWD* recurs twice in the original model. We express *AWD* as a type and apply two occurrences of it.

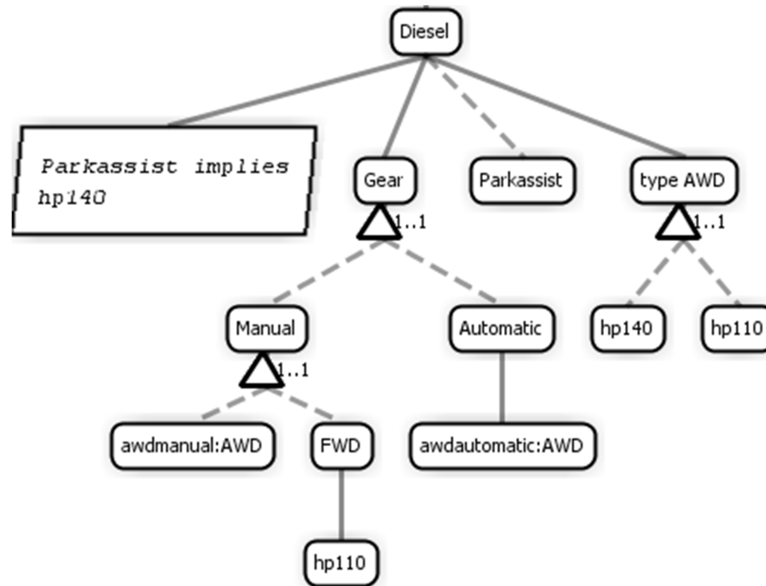

Fig. 6. The AWD variability type

The observant reader will have seen that replacing the two occurrences of *AWD* in Fig. 6 with replications of the type will not yield exactly the tree shown in Fig. 5 since for Automatic shift only the strong engine can be chosen. Such specialization should be expressed by a constraint associated with the occurrence.

We note that the type itself is defined on a level in the tree which encloses all of the occurrences. It is indeed not obvious where and how the type should be defined and we have shown here what was sufficient to cover the Autronica case.

In Fig. 6 *awdautomatic:AWD* is a ChoiceOccurrence, which represents an occurrence or instantiation of the *AWD* VType. A question is whether a ChoiceOccurrence can itself contain a tree structure below it since it is indeed a VNode? If there was a subtree with a ChoiceOccurrence as root, what would be the semantics of that tree acknowledging that the referred VType defines a tree, too? It is quite obvious that there must be some consistency between the occurrence tree and the corresponding VType tree. Intuitively, the occurrence tree should define a

narrowing of the VType tree. There are, however, some serious challenges with this. Firstly, to specify the narrowing rules syntactically is not trivial. Secondly, to assert that the narrowing rules are satisfied may not be tractable by the language tool. Thirdly, the narrowing structures may not be intuitive to the user. Therefore, we have decided that only constraints will be allowed to be used to further specify a choice occurrence. In our example case, the diagram in Fig. 6 would add a constraint below *awdautomatic:AWD* with exactly one target reference to *hp140* and thus the semantics would be the same as in Fig. 5.

Our example model has only Choices as VSpecs, but the Autronica system has multiple examples of elements that are sets rather than singular choices. Such decision sets that represent repeated decisions on the same position in the VSpec tree are described by VClassifiers. Similar to ChoiceOccurrences that are typed Choices, we have VClassOccurrences that are typed VClassifiers. We appreciate that VClassifiers are not VTypes even though they represent reuse in some sense, but sets are not types. A type may have no occurrences or several occurrences in different places in the VSpec tree.

5.5 Resolution Literals – Describing Subproducts

Once we have the VType with corresponding occurrences in the variability model, we may expect that there may be consequences of these changes in the associated resolution models and realization models.

What would be the VType counterpart in the resolution model?

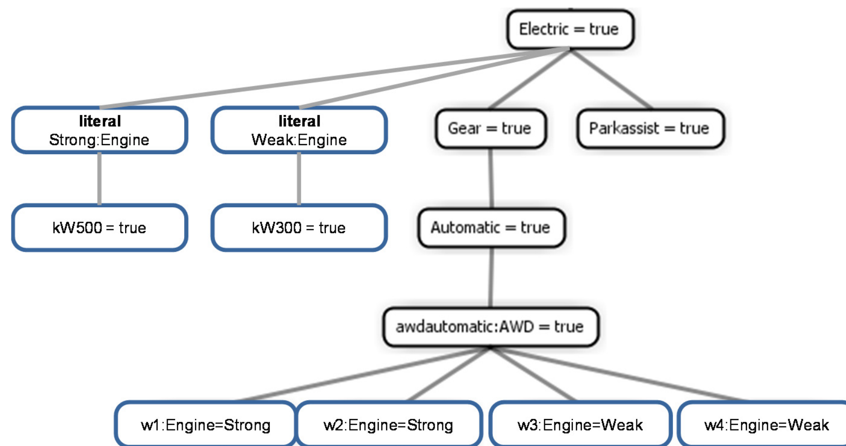

Fig. 7. Resolution literals

In Fig. 7, we show a resolution model of an imaginary electric car that has one engine for each wheel. We have defined two literals of the *Engine* type, one named *Strong* and one named *Weak*. The literals represent sub-products that have been fully resolved and named. In reality, it is often the case that there are named sub-products

that already exist and have product names. Thus such literals make the resolution models easier to read for the domain experts.

5.6 Staged Variation Points – Realizing Occurrences

Having seen that the VType has consequences for the resolution model, the next question is what consequences can be found in the realization model that describes the mapping between the variability model and the base model?

We have already reuse related to the realization layer since with fragment substitutions we can reuse replacement fragment types. Replacements represent general base model fragments that are cloned and inserted other places in the base model base.

Replacement fragment types do not correspond to VType directly and we find that with fragment substitutions as our main realization primitive we would need a hierarchical structure in the realization model to correspond to the hierarchy implied by occurrences of VTypes in the variability model. The "staged variation points" correspond closely with subtrees of the resolution model. They are not type symbols, but rather correspond to the expansion of occurrences (of VTypes and resolution literals).

In BVR (and CVL) variation points refer to a VSpec each. Materialization of a product is driven by the resolutions. They refer to VSpecs and trigger those variation points that refer to that same VSpec. A staged variation point refers to an occurrence of a VType.

The semantics of a staged variation point is to limit the universe of variation points from which to choose. The VSpec being materialized is an occurrence which refers to a VType. That VType has a definition containing a tree of VSpecs. The resolution element triggering the staged variation point has a subtree of resolution elements that can *only* trigger variation points contained in the staged variation point.

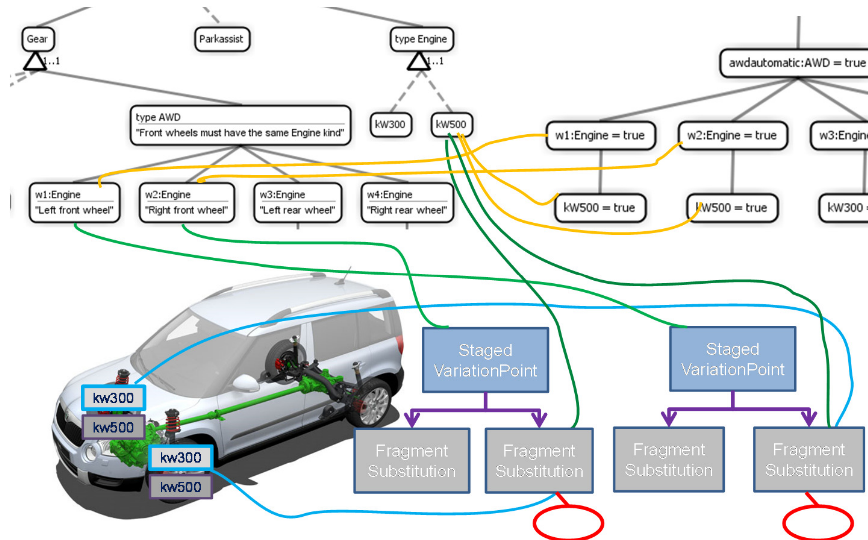

Fig. 8. Staged Variation Points example

In Fig. 8, we illustrate how staged variation points work. In the upper right, we have the resolution model and we will concentrate on resolutions of *w1* and *w2*. *w1* is resolved to true and the rightmost staged variation point refers the *w1:Engine* choice on the very left in the VSpec model indicated by the (green) line. Now since the *w1:Engine* has been chosen, we need to look into the *Engine* VType for what comes next, and the choice of the power of the engine comes next. For *w1* the resolution model indicates that *kw500* is chosen and this is also indicated by a (yellow) line from the resolution element to that of the VSpec model. The actual transformation of the base model is given by the variation points in the realization model, and we are now limited to the variation points enclosed by the staged variation point already found (the rightmost one). The rightmost fragment substitution of said staged variation point refers to the chosen *kw500* VSpec inside the *Engine* VType and thus this is the one that will be executed. The figure indicates that what it does is to remove the *kw300* option and leaving only the *kw500* engine option on the right wheel of the car.

In the very same way, we may follow the resolution of *w2* and we find that due to the staged variation point for *w2* is the leftmost one, a different fragment substitution referring the *kw500* of the *Engine* VType will be executed for *w2*, which is exactly what we need.

6 Discussion and Relations to Existing Work

Here we discuss the new mechanisms and why they have not appeared just like this before.

6.1 The Target

Introducing targets was motivated by how the VSpec tree structure can be used to visualize and define restrictions to decisions. The more the tree structure is used to define the restrictions, the more likely it is that there is a need to refer to the same target from different places in the tree.

Our example car in Fig. 4 can be described in another style as shown in Fig. 9 where the restrictions are given explicitly in constraints and the tree is very shallow. The two different styles, tree-oriented and constraint-oriented, can be used interchangeably and it may be personal preference as well as the actual variability that affects what style to choose. It is not in general the case that one style is easier or more comprehensible but constraints seem to need more familiarity with feature modeling [10].

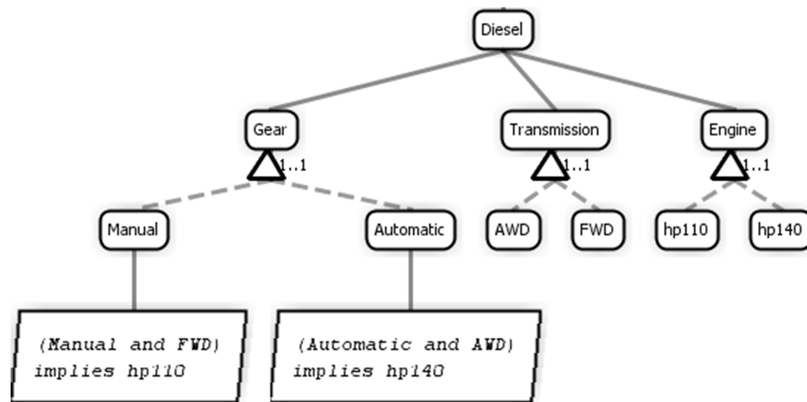

Fig. 9. The example car with explicit constraints

Given that a tree-oriented style is applied and there are duplications of target, why is this a novelty? It is a novelty because CVL does not have this concept and it is unclear whether other similar feature modeling notations support the distinction that we have named targets as distinguished from VSpecs (or features). Batory [11] and Czarnecki [12, 13] seem to solve duplication by renaming to uniqueness. The Atego OVM tool[4] implicitly forces the user into the style of using explicit constraints and thus circumvents the problem. OVM [4] does not contain the general feature models of FODA [5].

VSpecs are distinct decision points and every VSpec is in fact unique due to the tree path to the root. Targets are also unique, but for a different reason. Targets are unique since they represent some substance that is singular. This substance needs not be base-model specific, but it is often closely related to the base model. What makes this distinction essential is that explicit constraints talk about targets and not VSpecs.

---

[4] http://www.atego.com

In Fig. 5 we have a variability model which is properly satisfied by (*Parkassist, Manual, AWD, hp140*) and by (*Parkassist, Automatic, AWD, hp140*) showing that *hp140* may refer to any of two distinct VSpecs.

6.2 The Type and Its Consequences

Introducing a type concept to BVR should come as no surprise. As pointed out in [14] concepts for reuse and structuring normally come very early in the evolution of a language. Since the feature models have a fairly long history [5], it may be somewhat surprising that type concepts for subtrees have not been introduced before. A type concept was introduced in the MoSiS CVL [1], and this was fairly similar to the one we introduce to BVR. The CVL Revised Submission [3] has a set of concepts related to "configurable units" that are related to our suggested VType concept, but those concepts were intended mainly for sub-product lines of larger size. The concept was also much related to how variabilities are visible from the outside and the inside of a product line definition.

Other notations have not introduced type concepts and this may indicate that the suggested notations were not really seen as modeling languages, but more as illustrations. Another explanation may be that type concepts do introduce some complexity that imply having to deal with some challenges.

One challenge is related to notation. The type must be defined and then used in a different place. In the singular world definition and usage were the same. VTypes must appear somewhere. We have chosen to place them within the VSpec tree, but it would also be attractive to be able to define VTypes in separate diagrams. A VType in fact defines a product line in its own right. Our Engine VType implied in Fig. 7 could contain much more than only horse power choice.

In the modeling language Clafer, which has served as one of the inspiration sources of BVR, the type declarations must be on the topmost level [15], which in our example would have made no difference. Locally owned types, however, have been common in languages in the Simula/Algol tradition [16] for many years. The local ownership gives tighter encapsulation while the top ownership is semantically easier.

The usage occurrences refer to the type. How should this be depicted? We have chosen to use textual notation for this indicating the type following a colon. The colon is significant for showing that the element is indeed an occurrence of a VType.

Another challenge is related to how the VType and its occurrences are placed in the model at large. This has to do with what is often called scope or name space. We have defined that VTypes or VPackages (collections of VTypes) can be name spaces and thus occurrences of a VType *X* can only appear within the VType enclosing the definition of *X*, but VTypes may be nested. Similar to the discussion on targets, again names are significant because they designate something unique within a well-defined context.

Are targets and types related? Could we say that targets appearing in multiple VSpecs are in fact occurrences of a VType (named by the target name)? At first glance this may look promising, but they are conceptually different. The target is something invariant that the decisions mentioning it are talking about. A VType is a

pattern for reuse, a tree structure of decisions representing a subproduct line. There are cases where the two concepts will coincide, but they should be kept distinct. While VTypes are defined explicitly and separately, we have chosen to let targets be defined implicitly through the names of VSpecs.

CVL already recognized types as it had VariableType, which was quite elaborate and which also covered ReplacementFragmentType and ObjectType. Could VType be a specialized VariableType and the occurrences specialized variables? This may also be tempting, but variables are given values from the base model by the resolutions, while occurrences refer to patterns (VTypes) of the variability model.

6.3 The Note

The Note is about a significant element that has no direct significance in the language. Adding a note concept is an acknowledgement of the fact that there may very well be information that the user wants to associate closely with elements of the BVR model, but which is of no consequence to the BVR language or general BVR tooling.

Such additional information may be used for tracing, for expressing extra-functional properties or it may be pure comments. The text may be processed by proprietary tooling or by humans. Having no such mechanism made it necessary to accompany a CVL diagram with a textual description if it should be used by more than one person or more than one community.

Since variability modeling is oblivious to what varies, the Note can be more important than it might seem. The Note is where you can associate safety critical information with the variants and the possibilities. The Note is where you can contain traces to other models. The Note is where you can put requirements that are not connected to the variability model itself.

The Note will be significant for the tools doing analysis.

We foresee that once we have experimented with using notes in BVR, there will be recurring patterns of usage which may deserve special BVR constructs in the future, but at this point in time we find such constructs speculative.

6.4 The Reference

References in the BVR model are similar to what can be found in commercial tools like pure::variants[5]. A reference in the variability model is defined as a variable and as such it enhances the notion that variables hold base model values only. A *Vref* variable is resolved by a *VRefValueSpecification* where the pointers of the resolution model and the pointers of the variability model correspond in a commutative pattern.

Why are references necessary? They represent structure beyond the tree and this can represent dependencies that are hard to express transparently in explicit constraints.

In our motivation from the Autronica case our need for references came from describing an alternative product structure that overlaid the hierarchical physical

---

[5] http://www.pure-systems.de

structure of the configured system. We may say that our Autronica variability model is a very product-oriented (or base-oriented) variability model meaning that structures of the product was on purpose reflected in the variability model. The opposite would have been a property-oriented variability model where VSpecs would have represented more abstract choices such as "Focus on cost" vs. "Focus on response time".

7 Conclusions and Further Development

We have been motivated by needs of the use cases and found that the needs could be satisfied by introducing some fairly general new mechanisms. At the same time we have made the BVR language more compact than the original CVL language such that it serves a more focused purpose.

Our next step is to modify our CVL Tool Bundle to become a true BVR Tool Bundle to verify that the demo cases can more easily be expressed and maintained through the new language.

The future will probably see improvements along two development paths. One line of improvements will be related closely with needs arising from variability analysis techniques for safety critical systems. We suspect that the generic Note construct could be diversified into several specific language mechanisms associated with analysis techniques. This would migrate the insight from the analysis tools to the BVR language.

The second line of improvements will follow from general language needs. The VType concept should potentially form the basis for compact concepts of interface and derived decisions serving some of the same goals as the elaborated mechanisms around "configurable units" in CVL. We think this line of development will also include partial binding and default resolutions without introducing additional conceptual complexity.

Acknowledgements. This work has been done in the context of the ARTEMIS project VARIES with Grant agreement no: 295397.

References

1. Haugen, O., Møller-Pedersen, B., Oldevik, J., Olsen, G.K., Svendsen, A.: Adding Standardized Variability to Domain Specific Languages. In: Geppert, B., Pohl, K. (eds.) SPLC 2008, vol. 1, pp. 139–148. IEEE Computer Society, Limerick, Ireland (2008)
2. Haugen, O., Wasowski, A., Czarnecki, K.: CVL: common variability language. Proceedings of the 17th International Software Product Line Conference, pp. 277-277. ACM, Tokyo, Japan (2013)
3. OMG: Common Variability Language (CVL). Revised Submission, OMG (2012)

4. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering. Foundations, Principles and Techniques. Springer (2005)
5. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute, Carnegie Mellon University (1990)
6. Johansen, M.F., Haugen, Ø., Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. SPLC '12 Proceedings of the 16th International Software Product Line Conference - Volume 1, pp. 46–55. Association for Computing Machinery (ACM) (2012)
7. Johansen, M.F., Haugen, Ø., Fleurey, F., Eldegaard, A.G., Syversen, T.: Generating Better Partial Covering Arrays by Modeling Weights on Sub-product Lines. In: France, R.B., et al. (eds.): MODELS 2012. LNCS, vol. 7590, pp. 269–284. Springer-Verlag Berlin Heidelberg (2012)
8. Johansen, M.F.: Testing Product Lines of Industrial Size: Advancements in Combinatorial Interaction Testing. Ph.D. thesis, Department of Informatics, University of Oslo, Oslo (2013)
9. Berger, T., Stanciulescu, S., Ogaard, O., Haugen, O., Larsen, B., Wasowski, A.: To Connect or Not to Connect: Experiences from Modeling Topological Variability. In: SPLC 2014. ACM, to appear (2014)
10. Reinhartz-Berger, I., Figl, K., Haugen, Ø.: Comprehensibility of Feature Models: Experimenting with CVL. In: Juergen, D., Schulte, W. (eds) MODELS 2014. LNCS. Springer, to appear (2014)
11. Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005, LNCS vol. 3714, pp. 7–20. Springer, Rennes (2005)
12. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration Using Feature Models. Software Process Improvement and Practice, 10(2), 143–169 (2005)
13. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specifications. Software Process Improvement and Practice, 10(1), 7–29 (2005)
14. Haugen, O.: Domain-specific Languages and Standardization: Friends or Foes? In: Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S., Bettin, J. (eds.) Domain Engineering, pp. 159–186. Springer, Heidelberg (2013)
15. Bąk, K., Czarnecki, K., Wąsowski, A.: Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled. In: Malloy, B., Staab, S., van den Brand, M. (eds.) Software Language Engineering, LNCS vol. 6563, pp. 102-122. Springer Berlin Heidelberg (2011)
16. Birtwistle, G.M., Dahl, O.-J., Myhrhaug, B., Nygaard, K.: SIMULA BEGIN. Petrocelli/Charter, New York (1975)