Building BVR Models Better

Analysing and improving BVR and its tool for variability modelling

Martin Bernhard Øverbye Holøs

Master at the Department of Informatics

University of Oslo

03/08/2015

Building BVR Models Better

Analysing and improving BVR and its tool for variability modelling

Abstract

In this thesis, we aim to improve BVR and its tooling. In order to identify areas for improvement we have performed an analysis of the BVR language as well as an experiment that examines the error correcting process employed by BVR experts. We see this process as a lens under which to study the BVR language, tool and the experts' process. Our approach involves practical "in the wild" use of the tools functionality as well as insight into the experts workflow.

We have identified several areas of improvement regarding both the BVR language and tool, as well as work procedures. We have also made several suggestions for improvements to address these problem areas.

Foreword

Thanks to my supervisor Øystein Haugen, for all his help.
In addition, to Anatoly Vasilevskiy, for helping me with the technical work.

To my father, my brother-in-law and my friend Thomas, for your feedback.

And to my fiancé, for putting up with me while I work.

Index

Table of figures

1 Introduction

When a company produces either large quantities of customizable products (e.g. cars), or highly complex systems specialized for each customer (e.g. software systems), they can benefit from using Product Line Engineering (PLE) augmented with Variability Modelling. The benefits of this technique includes, amongst others, the reduction of overall development cost, enhancement of quality and a reduction in the time to market for future products (Böckle, Pohl, & Linden, 2005). This technique is used commonly enough in software development to warrant its own abbreviation SPLE (Software Product Line Engineering). It has even been stated that one of the primary ways of improving productivity in software development is through reuse (Boehm, 1999).

In recent years, SINTEF has developed the BVR (Base Variability Resolution models) language. BVR is an orthogonal, separate language for variability modelling allows for the automatic generation of customized products.

1.1 Research questions

In order to be relevant for application in actual production lines, it is desirable that the language and associated tools facilitates rapid construction of variability models with minimal risk of allowing impossible or meaningless varieties, e.g. cars without two gearboxes or no engine.

We have introduced the term Real World Applicable (RWA) to encompass varieties that the domain expert (in the running example used in this thesis, a car-manufacturer) sees as relevant to actually produce.

During our work described in chapter 3 Pre-experimental work, we have observed that creating a RWA model, and testing whether or not a model is RWA, are both difficult tasks. We believe that BVR-models are useful, but suspected that improvements that lessen the cognitive load of working with the models are possible, desirable, and perhaps necessary. We wanted to examine this further in the work reported herein.

The study was constructed to examine four research questions:

1. **Is BVR a good graphic language?** I.e. is BVR a cognitively effective visual notation?

2. **How do the experts work with the tool and language, how do these processes affect the outcome, and how are these processes affected by the tool?**

3. **Is the BVR tool a good tool?** I.e. does it facilitate work modellers must perform (including building, error correction and RWA checking), and does it help experts adopt good strategies?

4. **How can we make improvements to the BVR-tool, language, and processes to alleviate any shortcomings identified in question 1, 2, and 3, and can we prioritize the needed improvements using a standard categorization of errors?**

In order to answer these questions we have used a two-pronged approach. The first is a visual analysis of BVR using the design goals described by (Moody, 2010). These design goals are defined specifically to reduce the cognitive load of the modeller. The second part of our approach is a usability study using a combination of screen-capture and a questionnaire. In order to identify both the tool use and the processes, we chose to observe experts while they performed an error correction task. We further consider models as the result of a build process, and as such, we chose to include the impact of modelling styles in our experiment.

1.2 Structure of thesis

We have structured this thesis into seven chapters. The first three chapters are introductory chapters. Followed by two chapters dealing with the collection of and description of data answering regarding the first three question, and finally we propose improvements based on the results from the two previous chapters attempting to answer the last question.

The first chapter is introduction, which is the chapter you are currently reading.

The second chapter is the background material related to BVR, and contains a quick history of BVR and a detailed introduction into the BVR language and its constructs, includes information about Human Computer Interaction (HCI) and Usability research methods, as well as a chapter on the problem that sparked the research questions (the applicability problem).

The last chapter before we get into the real meat of the thesis is a chapter describing the work we, the author, did before we started the main experiment. That chapter contains details regarding the functionality we implemented before we ran the experiment, including functionality which was clearly used by the expert in the experiment, and which serves as a small confirmation of some of the theories used.

Chapter 4 is the visual analysis chapter, and contains both the method we used for the analysis and the results of the analysis. We have also decided to sort the chapter by terms so for each term there is a method description and a result. The results from this chapter are used both in chapter 5, to explain some of the observations, as well as in chapter 6 where they serve as both the main argument for an improvement and as additions to improvements based on chapter 5.

Chapter 5 contains the main experiment. In this chapter, we describe the experiment as well as report on the observations made.

Chapter 6 are a collection of suggested improvements based on both the experiment and the visual analysis.

Finally, chapter 7 contains our conclusions and suggestions for future work.

1.3 Limitations

In chapter 6 Proposed improvements, we have suggested several improvements to both the tool and the language, including the basis for implementing most of the. Due to both time limitations and the scope of this thesis we have note implemented the suggested improvements.

2 Background

2.1 The Evolution of BVR

In 1990 Feature Oriented Domain Analysis (FODA) was introduced by (K. C. Kang, Cohen, Hess, Novak, & Peterson, 1990) and as part of this method they introduced the first feature modelling approach. Since then several approaches have been introduced to the Feature Modelling family (e.g. FORM(K. Kang et al., 1998), (pure::variants) or KobrA(Atkinson, Bayer, & Muthig, 2000)). The variability model (see 2.2.2), or VSpec model, of CVL(Common Variability Language)(Haugen, 2012) is an evolution of FODA, and BVR(Base Variability Resolution)(Haugen, 2014) is a further evolution of CVL. The specific improvements of BVR over CVL is described by (Haugen & Øgård, 2014). For future reference it is important to note that BVR is a work in progress, and at the time this thesis was written, some of the improvements described in (Haugen & Øgård, 2014) had not yet been implemented, including Targets and Resolution Literals.

2.2 Introducing BVR

BVR(Haugen, 2014) is a tool to both help model the variability of a software product line (SPL), and to realize products from these models. Connecting BVR to a MOF-compliant((OMG)) base-model adds a layer of abstraction, which allows the modeller to work in the variability domain (the MOF specifications are the basis for an environment which allows models to be manipulated by different applications, amongst other things). This is possible because BVR is essentially a domain-specific language (DSL) devoted to the variability domain, which in turn permits BVR to describe most, if not all, SPL models. (Van Deursen, Klint, & Visser, 2000) :

Base-model

A base-model is a model in some MOF-compliant DSL. It is this model we change in the realization process described in Chapter 2.2.4 The realization model

A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

2.2.1 Why BVR

A BVR model consists of three parts which gives makes its usefulness threefold. The first part is the variability model (called a VSpec model in BVR). This model describes variations in a product line. In this capacity there are very few limits to the possible domains in which it can be implemented, be it a software product or a car customisation. Secondly, BVR can be used to model specific products from this VSpec model by assigning values to (resolving) the variation points, by building what is known as the Resolution model. Lastly, it is possible to connect a variability model to a MOF-based model. Connecting the VSpec model to the base-model, and creating a Resolution model gives BVR the power to make changes to the base-model through a process known as materialization, as described for CVL, the predecessor of BVR, by (Haugen, 2012). BVR is a work in progress and the current state of the implementation of the materialization process is currently unknown. This being said, the core concepts for the materialization process are described in chapter 2.2.4. In Figure 1 we see the metamodel describing the top levels of the BVR, while in Figure 2 we see the metamodel pertaining to the VSpec and VSpecResolution elements.

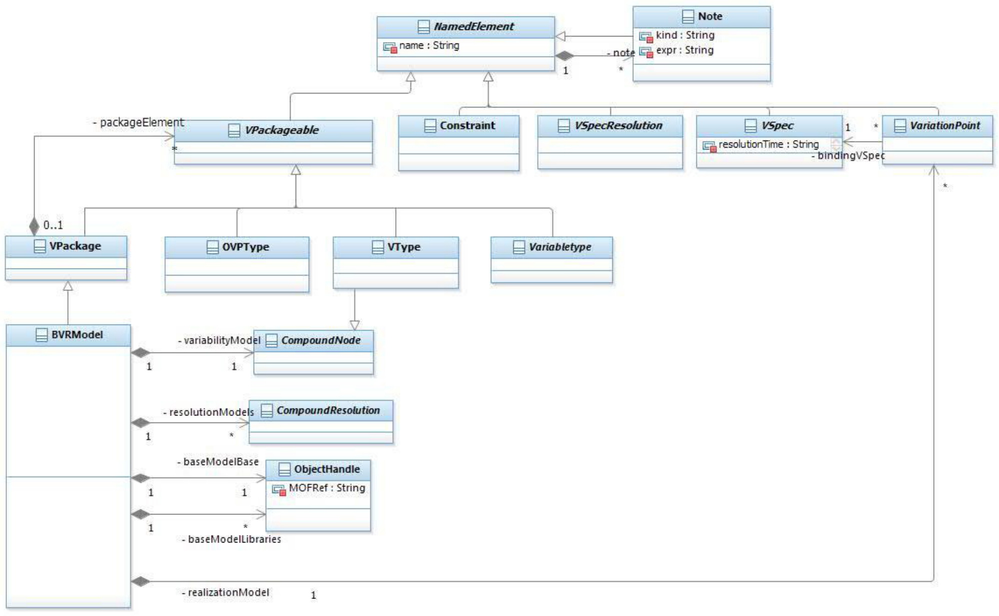


Figure 1 BVR top level(Haugen, 2014)

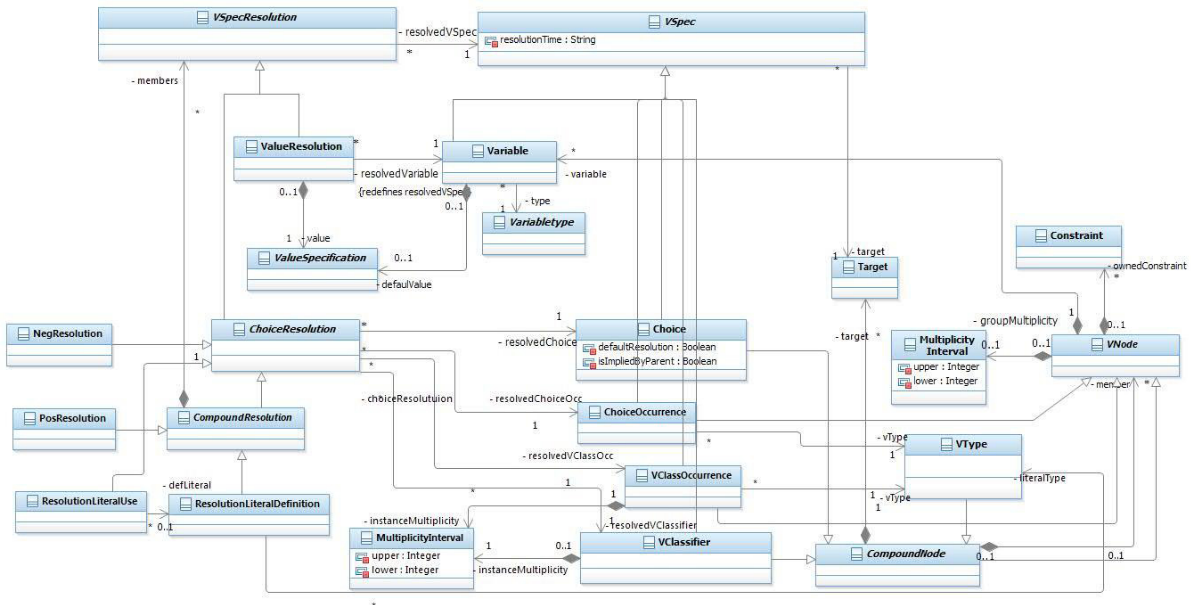


Figure 2 BVR VSpec and Resolution (Haugen, 2014)

In order to discuss the subject matter efficiently we will use a few abbreviations. First, we will use the term “group” or “group multiplicity” when we discuss the groupMultiplicity attribute. Secondly, we will use the term Resolution instead of VSpecResolution. For readability we will use the phrase “positive/negative Resolution” or “positive/negative Choice” when we talk about PosResoluiton and NegResolution.

2.2.2 The variability model

The first step in the BVR workflow process is creating the variability model. This model is built out of variation points called VSpecs, i.e. nodes in a hierarchy representing the different points of variation in the product line. The variability model corresponds to a feature model like the one described in (K. C. Kang et al., 1990)(FODA).

We will now introduce the meaning of the different terms used in the VSpec model.

A *Choice* is a Boolean variation point. The resolution (assigning of value in the resolution process) is either true or false, and represent the existence of the VSpec.

A *VClassifier* represents a variability point that have a variable number of instances, and always contains a multiplicity interval.

A *VType* is a separate sub-tree, which we in turn build like any other VSpec model. This construct, together with ChoiceOccurrences and VClassOccurrences, gives BVR both modularization capabilities, and internal reuse.

ChoiceOccurrences are Choices that contain a reference to a VType. This means that a ChoiceOccurrence will be resolved as a positive or negative resolution and its children will be the contents of the VType.

VClassOccurrences are VClassifiers that contain a reference to a VType. This means that they are resolved into one, or many resolutions, all of which contain the contents of the VType.

A *VNode* is a collection of all VSpecs except Variables (for more details see Figure 1).

A *group* is the attribute of VNode, which dictates the number of positive Choice resolutions the Resolution of the VSpec can have as its children in a valid resolution. This attribute is always a multiplicity interval. We usually make a distinction between the standard "no multiplicity" (the interval is from zero to many), "or" (one to many), "xor" (exactly one) and a custom multiplicity which the modeller sets him/her-self.

Comments are simply text fields in which the modeller can store temporary data or insert natural language explanations if needed.

Lastly, the *Constraints* used in BVR is a small version of OCL (Object Constraint Language)(Specification, 2006), called BCL (Basic Constraint Language). These Constraints are connected to a VNode and contain cross-model constraints which control the resolution of the model.

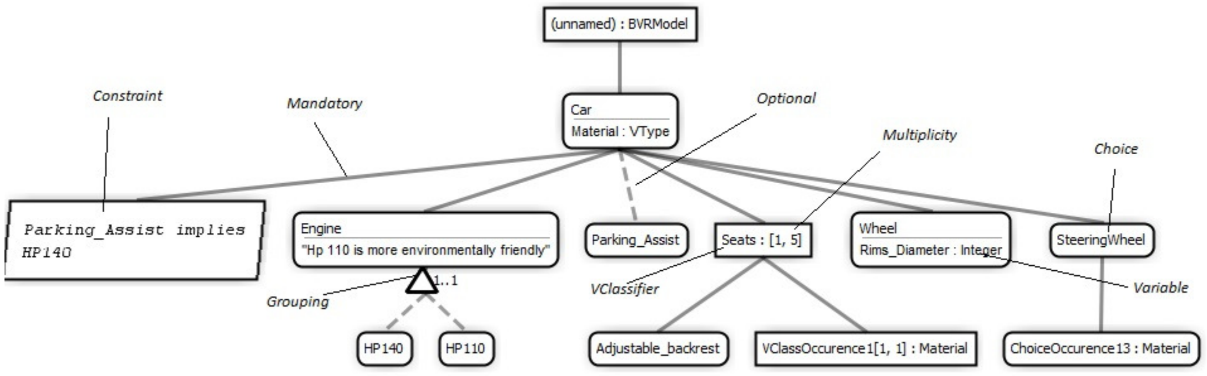


Figure 3 Example modelling of a car

2.2.3 The resolution model

A given BVR model can contain any number of resolution models. A resolution model is the resolution of, or allocation of values to, the nodes in the BVR variability model. This model corresponds to a configuration of a product from the product line. The type of resolution given to a VSpec depends, of course, on the VSpec it resolves.

Choices and variables are resolved as one might expect. A choice resolution must be resolved to either true or false (a choice must be made), while a variable must be resolved by assigning a value of the type given in the VSpec (Integer, string etc.).

VClassifiers on the other hand are always resolved to one or more positive Resolutions. The number of instances are dependent on the VClassifier's multiplicity interval.

The final two (ChoiceOccurrences and VClassOccurrences) reference a VType, and will resolve into a positive Choice resolution, whose children will be the resolution of the VSpecs referred to in the VType.

2.2.4 The realization model

The realizationModel consists of the variation points used to represent the realization of the variabilityModel given the resolutionModel. The variation points define the changes done with the base model to fulfill the configurations specified. (Haugen, 2014 p.13)

In order to create a realized system automatically with BVR we must first connect the VSpecs to variation points in the base-model. We can then create a resolution of the VSpec and begin the materialization process.

Materialization is the idea of using handles to a base-model, and possibly compatible model pieces in a library, in order to make changes to the model. There are three ways to materialize a base model:

1. The minimal baseline model approach. With a library of available features, performing additive alterations to the base model during the materialization.

2. With a maximal baseline model. Using a subtractive method where all negative choices are removed.

3. Using a combination of the previous two. This final method can be especially effective with prior knowledge probable features.

Although this description seems to imply that there is no way to insert a part of the model in between two others. Not so. BVR allows for so called “fragment substitution”, i.e. fragments in the base model (the placement fragment) that can be replaced by a fragment from a library (the replacement fragment).

One of the requirements of the materialization process is reflected in the idea of information hiding. For those readers familiar with programming this concept is similar to the interface concept used in JAVA. That is, any part of the base model to be replaced by another due to a choice in the resolution must have an identical connection, see Figure 4 Fragment substitution

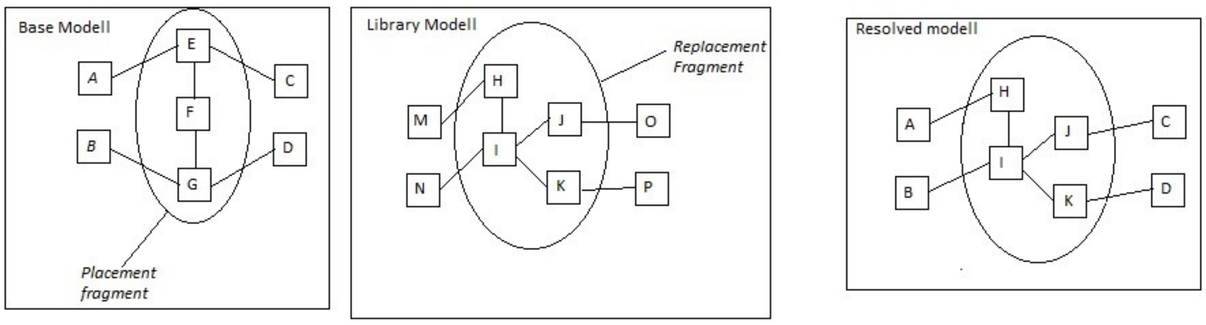


Figure 4 Fragment substitution

2.3 Methodologies in Human Computer Interactions and usability

In order to make a reasonable choice regarding the use of research method we have done some research into the fields of HCI (Human Computer Interaction) and usability, and found and evaluated several research methods.

The criteria we set for research methods where:

- It must not require hardware we do not have access to.
- It must not require training the author does not have access to.
- The data collected must be manageable and useful.
- The chosen method must allow insight into both the use of the tool and the process.

- The data must be useful even with relatively few participants.
- It must have a minimal affect the participants.

As we stated in the introduction, we decided that we wished to study experts performing an error-correction task in an environment that was as natural as possible. This, unfortunately, means that we require the time of experts, which is a scarce resource.

(Pohl & Scholz, 2014) Discusses eye tracking, thinking aloud, and log files, in addition to mentioning questionnaire and interviews. We will user their summation of the three first methodologies as the basis when discussing the usefulness of these methods in regards to the restrictions our stated goal.

Eye tracking

Tracking what a user is looking at is in most commercial eye trackers typically done with the help of an IR camera mounter next to (or beneath) a display monitor, using the corneal—reflection/pupil-centre method (Jacob & Karn, 2003). On one hand, eye tracking provides unobtrusive gathering of data. On the other hand this data is labour intensive to extract, difficult to interpret and technically difficult to collect(Jacob & Karn, 2003). The usage of eye tracking is also better suited to improve specific UI elements, and effective analysis requires precisely defined tasks (e.g. “add this Choice”, “edit that Constraint”). Considering both the availability of eye tracking equipment, and our wish to study the “real world” use of BVR, we decided against eye tracking.

Thinking aloud

The process of thinking aloud involves exactly what one would expect. You ask the participants to use the system while continuously speaking their thought aloud. This method gives insight into the cognitive process of the expert during the experiment, which could be useful, but unfortunately, it is also disruptive and can lead to unnatural behaviour. Since we early on decided that we wanted to study the tool as “in the wild” as possible we did not use this method.

Log files

Logging the user interaction for mathematical analysis is again an unobtrusive data gathering method. In addition, it is relatively quick to implement, and it reflects actual behaviour. The only disadvantage mentioned is the difficulty to interpret the data due to lack of context. In addition to this disadvantage, this method would be less useful due to the restricted number of data points (i.e. the number of experts), which is why we decided against the use of this method.

Questionnaires

Questionnaires as defined by (Kirakowski, 2000) are "questionnaire is a *method* for the *elicitation,* and *recording,* and *collecting* of information".

Questionnaires are relatively cost-efficient to administer, but difficult to develop. This makes questionnaires useful as a main source of research when you can draw upon many participants. Questionnaires also require that the author knows ahead of time what problems might be of interest.

Due to the low number of available participants and without knowledge of what might be the most important aspects of the improvement process, we chose to avoid this as our main method. As will be described in 2.3 Methodologies in Human Computer Interactions and usability we did use questionnaires to supplement the main experiment.

Interviews

While interviews are a rewarding form of usability measurement (Kuter & Yilmaz, 2001), it is also one of the most challenging. It also requires the interviewer to have training in interviewing techniques in order to be truly effective. Finally interviews are quite time consuming.

Since the experimenter does not have the required training to be effective, the experts have limited time, and the experimenter is one of the experts we have decided against the use of interviews.

Diaries

In HCI research diaries are mostly what you would expect, namely a self-reported collection of data collected over a relatively long period (typically one to two weeks). The diary can, on a broad scale be classified into either a Feedback Diary in which the user reports in the diary when a certain event or threshold has been reached, or a Elicitation diary, in which the user is encouraged to make entries when they are encounter events which they find meaningful(Carter & Mankoff, 2005). In this context, Elicitation diaries are meant to be a part of synchronous communication (e.g. an interview) while feedback studies would involve asynchronous communication (e.g. a questionnaire).

(Lazar, Feng, & Hochheiser, 2010) sums up the strengths and weaknesses of using diaries as a research method nicely, and we will now present the relevant points. They are good for non-controlled settings, for understanding the why of technology interaction, and good for recording occurrences with less personal interpretation of data as compared to other self-reporting methods. On the weakness side there are time taken to follow, the possible lack of participant introspection, and lower accuracy than laboratory settings. In our setting, the use of diaries present three problems. First, BVR would have to be in daily use by the experts, which it is not. The second, is the lack of expert time available. The last problem is that this research method would only highlight problems, which either the experimenter foresaw or the experts noticed.

Video

Recording the experiment participant and the screen through video and audio, and encoding the behaviour in a behaviour grid is according to (Bastien, 2010) a common way to test usability. In the method (Bastien, 2010) describes the experimenter determines frequency and duration of specific behaviour, as well as time taken to finish defined tasks and other meaningful measurements. (Bastien, 2010) further claims this requires both specific hardware and software, in addition to being highly time consuming. For our experiment, this technique is useful as it enables us to record both tool use and procedures employed by the experts. It does however, have a few vital drawbacks. First, recording the expert and the screen with sufficient clarity can be difficult without a professional setup. Second, the recording of the experts themselves would be of little use, as we have no training if facial analysis and we have decided against using the "speaking aloud" method. We did however end up using a variant of this technique, namely screen capture. This method allows the capture of the screen with a high degree of clarity without the distraction of additional (and useless) information.

2.4 The applicability problem

When we discuss feature models there are two aspects of the model that are of particular interest. The first is the correctness of the model, i.e. whether or not the model contains formal errors such as dead features and full-mandatory features.

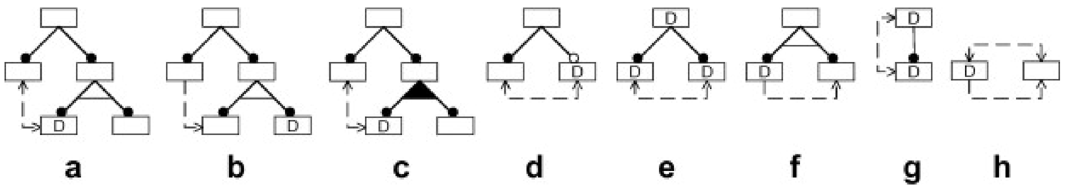


Figure 5 Dead Features (Trinidad, Benavides, Durán, Ruiz-Cortés, & Toro, 2008)

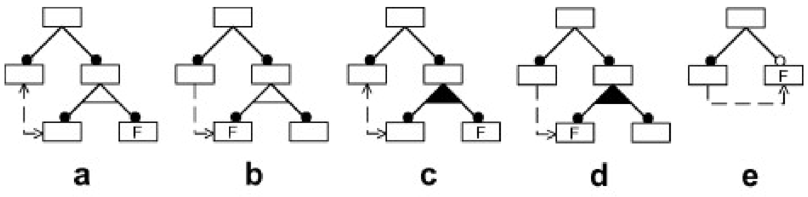


Figure 6 Full-mandatory features (Trinidad et al., 2008)

The second aspect is the real word applicability (RWA) of the model, i.e. whether or not the model contains all the intended features, and no other.

While we have yet to find research pertaining directly to the second problem, there are several interesting papers working on the first. In order to introduce agile techniques to SPL (Trinidad et al., 2008), have implemented a tool which checks Feature models (FMs) for errors. These errors are the same as those referred to in (von der Massen & Lichter, 2004) as those creating inconsistent models. This solution reduces the feature model to Constraints Satisfaction Problems (CSP). A different method for validating feature models is proposed by (Zhang), this method uses feature relationship propagation, and is reported by the author to be more efficient than the CSP method.

3 Pre-experimental work

When we started the work on this thesis, we were tasked with improving the resolution editor. This work, which is described in detail later in this chapter, was in part derived from observations made by one of the other experts. This work is also the author's qualification for expert status with BVR and the BVR tooling.

Since this chapter will contain information regarding the implementation of tool improvements, we must also give some information on the choices regarding technology. The BVR-project utilizes the Eclipse Modelling Framework (EMF)(Eclipse-Foundation), and the BVR metamodel is modelled and implemented with Ecore (which is the metamodel core of EMF). The graphic implementation of the tool is based on Swing (Oracle) which is not thread-safe (i.e. not guaranteed to safely execute shared data safely when multiple threads are running("Thread safety,")). The BVR tool is designed to allow multiple editors to work on the same model at the same time, in order to deal with this the BVR tool uses commands in the EMF transactional domain, which is used to track and synchronize changes to the model across the editors. The tool and code is available at https://github.com/SINTEF-9012/bvr.

Resolve tree

When we started on the improvements for the resolution tool, the modeller had to add one resolution at the time (see Figure 7).

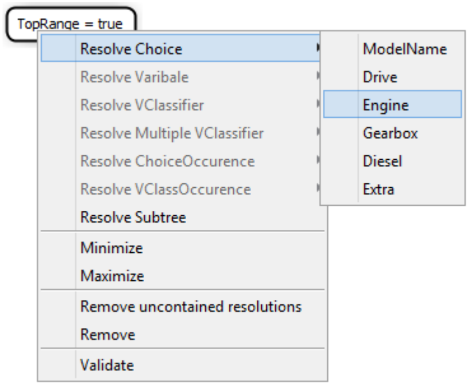


Figure 7 Adding Resolution manually

This way of resolving models was extremely time-consuming and required either detailed knowledge of the VSpec or both (VSpec and resolution) models on screen at the same time, in order to resolve for choices lower in the tree. We solved this by automatically building a resolution model that contain positive resolution and variables with default values. To be specific the algorithm adds:

- Choices as positive Resolutions.
- ChoiceOccurrences as positive resolutions with their contents added as a sub tree (see Figure 8, Figure 9, Figure 10).
- VClassifiers as a number of positive resolutions according to the lower multiplicity.
- VClassOccurrences as several resolutions with subtree from type.
- Variables as a resolved variable with a default value according to the type.

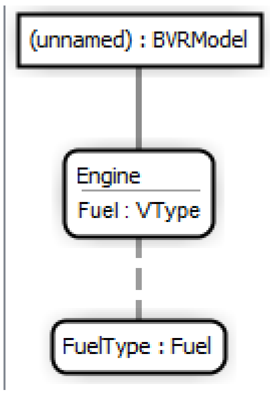


Figure 8 Type example

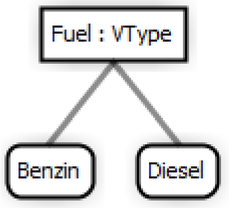


Figure 9 Type model

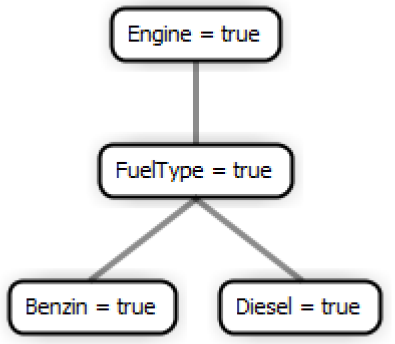


Figure 10 Type example resolved

We later expanded the auto-build function to work with sub-trees, which can be used to incorporate additions made in the VSpec model.

This, in turn, led to the creation of a “remove unconstrained resolutions” function, which iterates the tree and removes resolutions that no longer have corresponding VSpecs.

These functionalities were made using a recursive tree-iterator and a clone-algorithm. When we developed these functionalities, we found an interesting problem with the transactional domain, namely that even with “batch execution” the execution time was high (several seconds for a single command) even for small models due to the overhead of the transactional commands. The solution was to make changes to a clone of the subtree we wished to edit (e.g. add resolutions for unresolved VSpecs), which we could then edit outside the transactional domain. We then had to delete the old root node of the subtree and transactionally add the root of the new tree to the parent of the old node.

Auxiliary notation and grouping errors

The BVR metamodel defines Groups and Constraints as part of the VSpec model. While ChoiceResolutions reflect Choices, ChoiceOccurrences, VClassifiers, VClassOccurrences, and variable resolutions reflect Variables, there are no reflections of Groups or Constraints in the Resolution model even though these constraints greatly affect what is a valid resolution. We decided that the best cause of action was to introduce Grouping and Constraints as auxiliary notation for the resolution view. We implemented this functionality by checking the multiplicity interval of the resolved VSpec and showing the grouping symbol or checking the resolved VSpecs Constraints (which is its own list).

After implementing the auxiliary notation, we wanted to reduce the cognitive load of resolving models correctly. We did this by automatically checking whether the multiplicity constraint was fulfilled, and colour coded the grouping if there was an error. We later realized this solution was not sufficient, since it would not be useful for the colour blind, and we would suggest adding texture or making similar adjustments to the error-symbol.

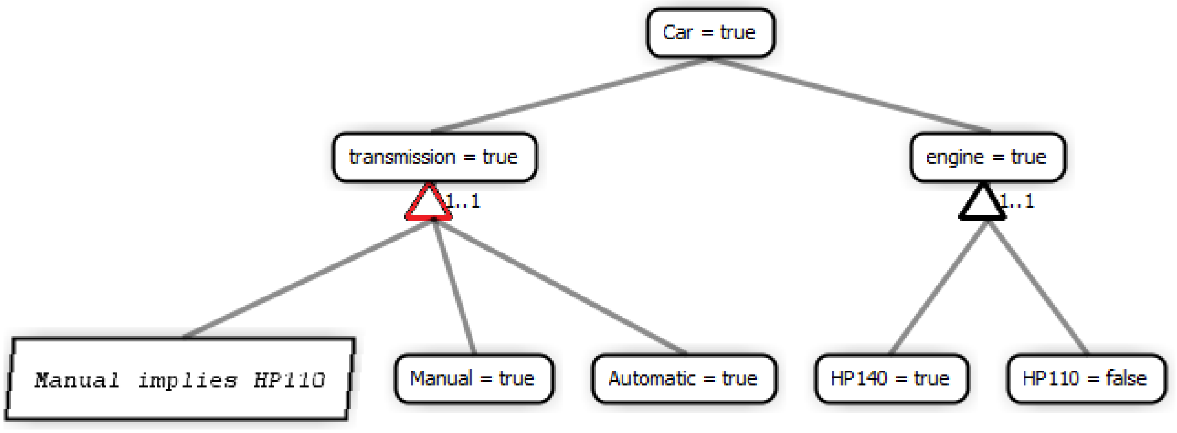


Figure 11 Example of the auxiliary notation

Minor and deprecated changes

In the earlier versions of BVR, both positive and negative resolutions have children, which left the BVR resolution model lacking in Complexity Management. Specifically, it meant that large parts of a resolution model could “exist without existing”, i.e. its parent choice (or higher ancestor) was negative, but the choice remained in the model. We therefore implemented a “stripped” functionality that hid the children of negative resolutions.

In addition, we implemented undo/redo functionality with shortcuts, but this was only a minor change since the functionality is implemented in the EMF transactional library.

Lastly, we have been part of the work of migrating from an observer-based architecture to an MVC focused architecture, in which the model logic is implemented to be UI-agnostic (i.e. the controller translates from UI elements to model elements. The result of this migration is a tool where we can implement a new UI without changing the business logic. This change in architecture is in part to prepare for a change in UI implementation.

4 Visual assessment of BVR

Since we wish to improve not only the BVR-tool but also BVR itself, we have decided to analyse BVR according to the theory described by.

4.1 Method

In order to assess the cognitive effectiveness of BVR (q.1), we will evaluate BVR in accordance with the design principles put forth by (Moody, 2010). In this assessment, we will also discuss some of the functionalities of the BVR tool, as it has direct impact on the cognitive effectiveness of the language. These design principles are Semiotic Clarity, Perceptual Discriminability, Semantic Transparency, Complexity Management, Cognitive Integration, Visual Expressiveness, Dual Coding, Graphic Economy, and Cognitive Fit.

4.2 The Semiotic Clarity Principle

The Semiotic Clarity refers to the relationship between the graphic symbols and semantic constructs of the language. This principle states that there should be a 1:1 relationship between these. If this is not the case, there are four possibilities:

Symbol Redundancy: multiple symbols for one semantic construct.

Symbol Overload: a symbol represents two or more semantic constructs.

Symbol Excess: there exists symbols that does not represent a semantic construct.

Symbol Deficit: there exist semantic constructs that are not represented by a symbol.

Results

BVR has symbol overload problems caused by two sets of homonyms (identical graphic symbols). These sets are VClassOccurrences (Figure 12) and VClassifiers (Figure 13); and Choice (Figure 14), ChoiceOccurrence (Figure 15), ChoiceResolutions (Figure 16) and VariableResolution (Figure 18). In addition, there is a debatable symbol redundancy, wherein a VClassifier with a multiplicity of one, is always resolved in the same manner as a mandatory choice, while technically they are different Semantic constructs.

The Symbol overload problem is characterised as the worst of the semiotic clarity problems as it leads to ambiguity and causes the potential for misinterpretations.

4.3 The Perceptual Discriminability Principle

The problems with Semiotic Clarity also tie into the Perceptual Discriminability of the graphical constructs. This principle deals with the ease and accuracy with which graphical symbols can be distinguished from one another, and is primarily measured in visual distance (i.e. the number and intensity of visual differences).

Results

The first problem with BVR and Perceptual Discriminability is its almost complete lack of colour and texture. While it is necessary to avoid encoding information in colour only (so the tool/language is usable for those with colour vision deficiency), it is useful as a secondary encoder to increase the visual distance.

The second Discriminability problem is that BVR has an abundance of quadrilaterals. This means that the visual distance between VClassifiers and VClassOccurrences is zero as they are both defined as squares (Figure 12 and Figure 13).

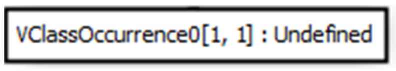


Figure 12 VClassOccurrence

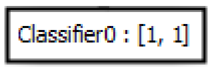


Figure 13 VClassifier

The visual distance between the classifiers (VClassifiers and VClassOccurrences) and the choices (Choices (Figure 14), ChoiceOccurrences (Figure 15), Positive (Figure 16) and Negative (Figure 17) Resolutions) and Variable resolutions (Figure 18) is close to zero and the difference between the choices and the Variable resolution is zero as they are all rounded squares.

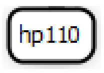


Figure 14 Choice

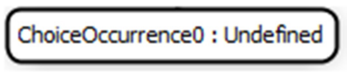


Figure 15 ChoiceOccurrence

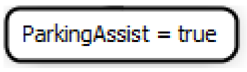


Figure 16 Positive Resolution

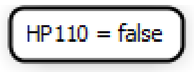


Figure 17 Negative Resolution

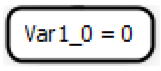


Figure 18 Resolved Variable

The final quadrilateral is the Constraints, which are parallelograms (Figure 19), and again the visual distance is low.

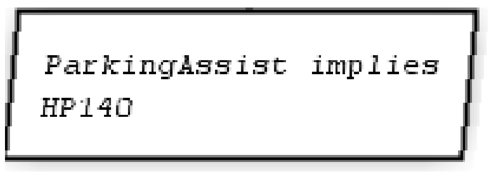


Figure 19 Constraint

Which leaves us with the only non-quadrilaterals being the grouping triangle (Figure 20, Figure 21, Figure 22, and Figure 23), and the variable and type textual additions (

Figure 24) (which in this context have zero visual distance). While the visual distance between the before-mentioned quadrilaterals and the groups are higher than the other distances (since they belong to a different shape family) the distance is still not overwhelming, since the differences are only in shape and size.

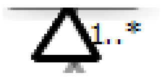


Figure 20 OR group

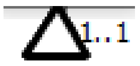


Figure 21 XOR group

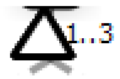


Figure 22 Custom group

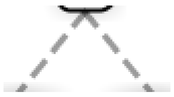


Figure 23 Ungrouped (0..)*

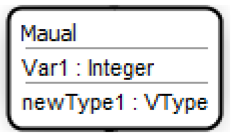


Figure 24 Variable and type in VSpec

The final problem related to Perceptual Discriminability is the visual distance between OR (Figure 20), XOR (Figure 21), custom grouped (Figure 22) and ungrouped (Figure 23) choices.

While BVR has problems in this area, it is also an improvement over other feature modelling languages, like FODA (K. C. Kang et al., 1990). FODA uses a small a small white dot to indicate optionality (see Figure 25), while BVR uses dashed/solid lines (see Figure 26).

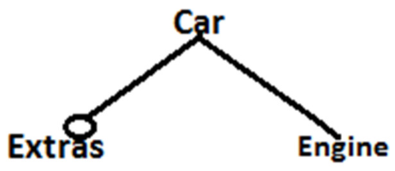


Figure 25 Optional and Mandatory relationships in FODA

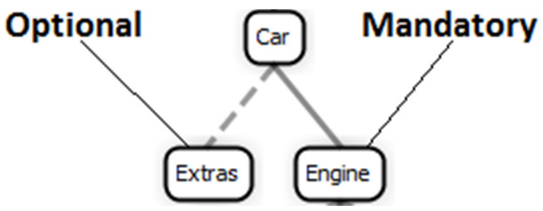


Figure 26 Optional and mandatory relationships in BVR

4.4 The Semantic Transparency Principle

Semantic Transparency pertains to the “intuitiveness” of the language. This concept divides naturally into two parts. The first is the Perceptual Resemblance i.e. “can a symbol’s meaning be inferred from its shape?”.

The second part of Semantic Transparency is Semantically Transparent Relationships. This principle deals with the ease of which someone can understand the relationship between visual elements.

Results

As BVR deals with an abstract space (variability), building in Perceptual Resemblance would be hard, and BVR is currently semantically opaque on this note.

The second half, namely Semantically Transparent Relationships, is on many levels well handled by BVR. Since all VSpec/Resolution, diagrams are tree structures, which seem well suited for the type of information modelled. Since it is possible to model systems in BVR in a variety of different styles, the two main groups of which are hierarchical or flat, it is important to note the sentiment from (Gurr, 1999):

The effectiveness of a representation is to a significant extent determined by how closely the semantics of the representation resembles that which it represents.(p 325)

While this does not mean that one style is always superior to the other, it means that, for the kind of models that are in our experience the most common, the hierarchical structure is preferable.

4.5 The Complexity Management Principle

Defined as the effectiveness of mechanisms implemented to deal with diagrammatic complexity (number of elements in a single diagram). The methods (Moody, 2010) suggest for dealing with complexity are modularisation, and Hierarchy.

Results

BVR handles diagrammatic Complexity Management (number of elements in a diagram) through three main mechanisms. The BVR tool implements a minimize functionality which lets the modeller hide subtrees, while BVR defines VTypes which adds a layer of abstraction. We believe these mechanisms can be utilized during the modelling procedure to produce diagrams with relatively low complexity even for complex systems. The last feature is also a

BVR language feature, namely that negative resolutions do not contain child resolution. The usefulness of the first two mechanisms however do depend highly on the modelling style used. Figure 27 shows P14 after correction, and is an example of a hierarchical style model. Figure 28 shows the same model with luxury and manual minimized.

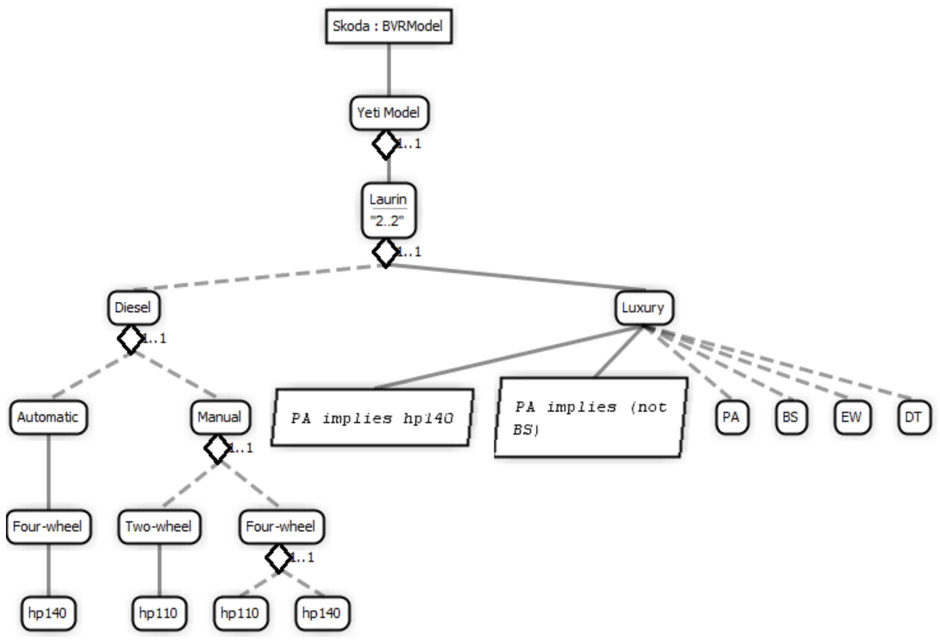


Figure 27 P15

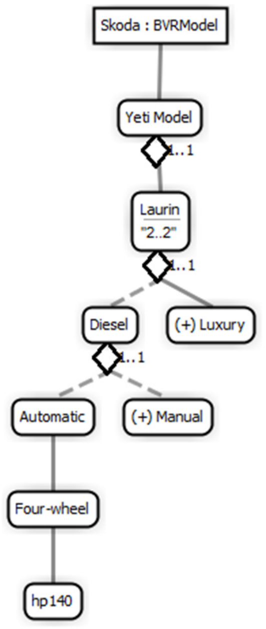


Figure 28 P15 minimized

On the other hand, a flat structure such as Figure 29 will maintain a greater deal of complexity with the same number of nodes (extras and engine) minimized which we can observe in

Figure 30.

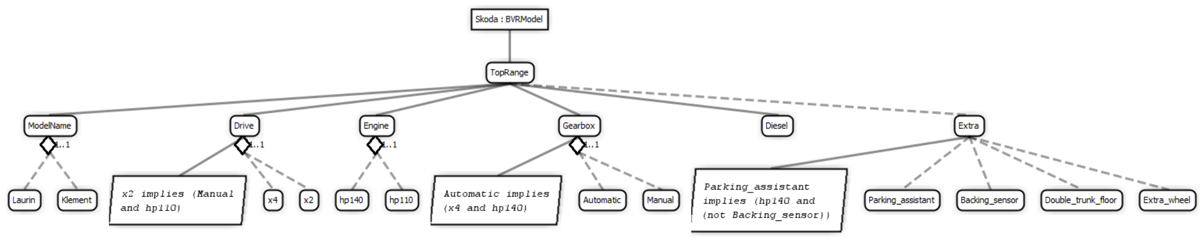


Figure 29 P14M

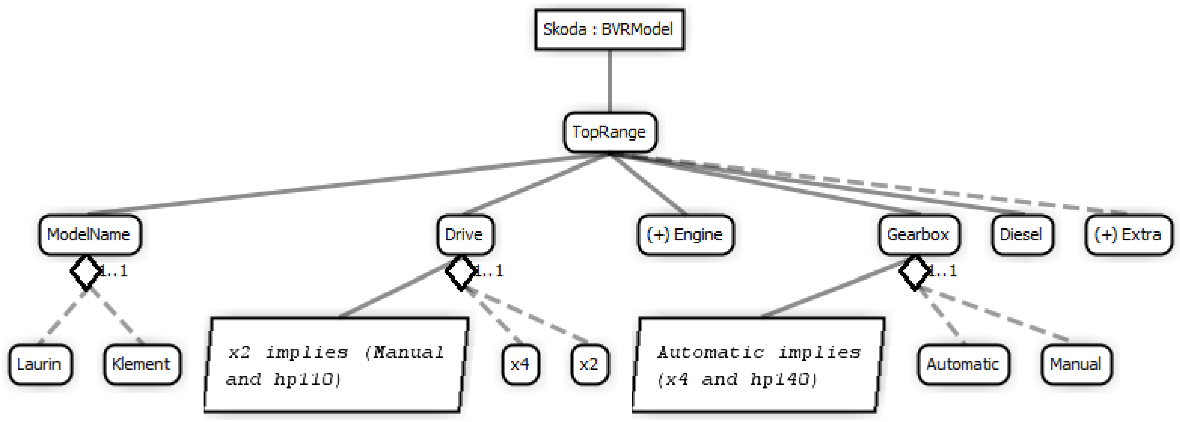


Figure 30 P14M minimized

The Cognitive Integration Principle i.e. integration of data from other diagrams representing the same system, can in many ways run counter to the Complexity Management Principle as it will often increase the number of elements represented on the same diagram. The Cognitive Integration Principle is divided into two parts. The first is Conceptual Integration, which deals with mechanisms to ease the modeller's cognitive load when trying to assemble a coherent picture of the system. The second is Perceptual Integration, which deals with inter-diagram navigation. In BVR, there are two different kinds of diagram interactions, which we have to analyse. The first is the interaction between the different parts of the model (VSpec, Resolution, and Realization) which we will call model-interaction, these interactions are heterogeneous, i.e. they contain models of different types. The second is the interaction between VTypes and each other and the base VSpec model, which we will call type-interaction, which are homogenous, i.e. they contain only the same type of elements.

When reading (Moody, 2010) it is clear that some of the definitions used do not apply to BVR, due to the interaction of the VSpec and Resolution diagrams. Creating a summary diagram between VSpec and Resolutions, for instance, would not be very useful. A summary diagram could be useful, however when dealing with types. (Moody, 2010) further introduces Contextualisation, which is the idea of introducing parts of the diagrams that are connected to the diagram one is currently focusing on. BVR does not include any Cognitive Integration other than the auxiliary notations, which we describe in Chapter 3 Pre-experimental work, and which is related to, but not quite, Contextualization.

4.6 The Visual Expressiveness Principle

The Visual Expressiveness refers to the breadth and range of the visual variables used in a language. Not to be confused with The Perceptual Discriminability Principle, which we measure pair-wise, Visual Expressiveness measures the total visual variability of a language.

Results

When we discussed visual distance, we found that there are limited differences, even between the symbols which are the most different. It is therefore no surprise that we considered the Visual Expressiveness of BVR to be relatively low. On the other hand, this leads to a language that has many free visual variables, which can be utilised by the tool. In its current form, BVR uses a hierarchical representation, and as a result horizontal and vertical positioning cannot be used further to encode information. In addition, size is adapted to the content of the nodes, which leaves colour, texture, shape, orientation, and brightness. Further, colour blindness means that colour and texture should not be used independently of each other, which leads us to believe that these variables should be left as free variables together with brightness.

4.7 Principle of Dual Coding

In Perceptual Discriminability and Visual Expressiveness, we have regarded text as having zero visual distance. That, however, does not mean that text has no impact on the cognitive load from reading a diagram. Quite to the contrary, the Principle of Dual Coding states that text and graphics conveys information more efficiently when presented together then when they occur separately.

Results

In BVR, there is little use of dual coding. There is some in the difference between “no group” and “OR group”, but since there is no difference in the symbol for “XOR group” or “Custom group” this coding is effectively useless. This means that the introduction of a new symbol for XOR should not lead to the removal of the “1..1” notation. We call the encoding of information both graphically and as text “hybrid symbols”.

4.8 Graphic Economy

Almost the polar opposite of Visual Expressiveness, and not to be confused with Complexity Management, we find Graphic Economy. This principle states that the language needs have a cognitively manageable number of different graphic symbols.

Results

In BVR, this is not a big problem, since there are only a small number of language constructs. This is also one of the principles where the expert-novice difference is the highest, since novices must actively keep the meaning of different structures in working memory. (Miller, 1994), claims that the human mind can discriminate between about six different categories. The BVR VSpec model currently contains 10 sematic constructs, but only four graphic constructs (square, rounded square, parallelogram, and triangle).

4.9 Cognitive Fit

The final principle is the principle of Cognitive Fit, which states that different visual dialects are suitable for both different audiences (expert-novice difference especially) and different situations (e.g. working on a whiteboard vs. in a program).

Results

BVR is an example of a language, which exhibits visual monolinguism (i.e. only one view). Since Graphic Economy problems are more pronounced when the modeller is a novice, one possible dialect could include different symbols for different groups (OR, XOR etc.).

4.10 The reasons for the current version

After speaking with one of the creators of CVL/BVR (Øystein Haugen), we learned a few things regarding the choices made with respect to the graphical choices made. The first pertains to the graphic design of Choice and ChoiceResolution. Both constructs are rounded rectangles, and signify a reflection of meaning. I.e. the graphical representation of a ChoiceResolution shows that the modeller is resolving a choice. The second graphic choice we discussed was the choice of quadrilaterals as the main group of graphic constructs. This choice is a reflection of the historical choices made by (K. C. Kang et al., 1990), (pure::variants) and others(see Figure 31 and Figure 32). The third insight we gained regarded the choice of structure, which is also influenced by history. There is no absolute reason to represent BVR models as hierarchical diagrams however, although the decision is backed by (Winn, 1990), and it could just as well be represented as, for instance, a matrix.

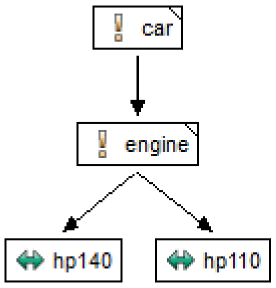


Figure 31 pure::variants graphic view

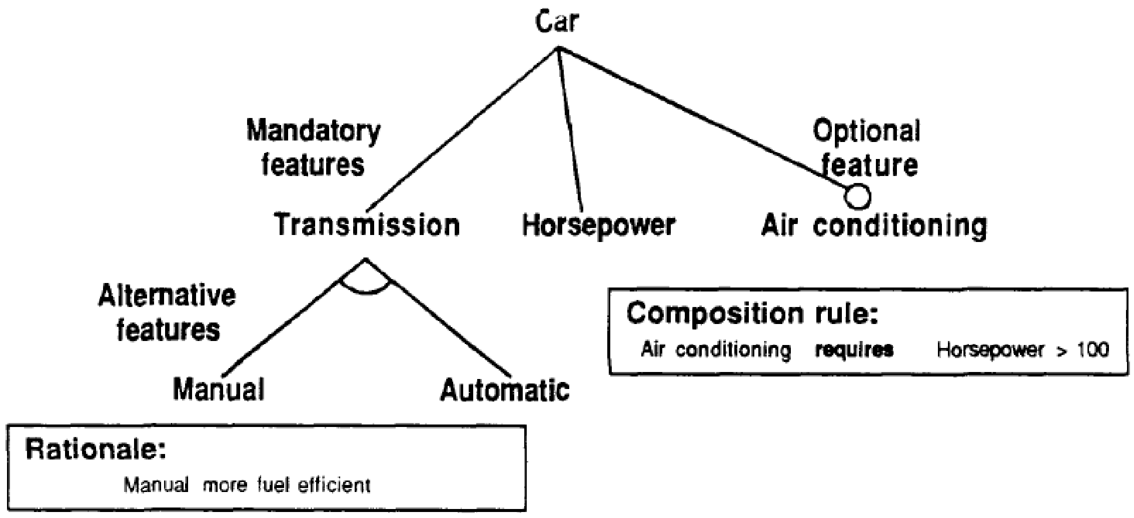


Figure 32 FODA Feature model Example(K. C. Kang et al., 1990)

5 Expert experience

In our effort to identify whether or not the BVR tool is a good tool (q.3), and examine the processes involved in expert BVR use (q.2), we decided to test the BVR tool and language in a situation which was as close to "in the wild" as possible. We therefore designed an experimental procedure that would let experts use the tooling functionality at their own discretion while still letting us review their actions later. After reviewing the methods we discussed in chapter 2.3, we decided to use screen capture together with an informal, free text questionnaire. The choice of screen capture is an adaptation of the video method, which ensures good video of the screen, while avoiding the collection of unnecessary data.

5.1 Method

To study expert's use of BVR and its tool, we decided to test the BVR tool and language in a situation, which was as close to "in the wild" as possible, using screen captures, and a questionnaire. Alternative methods are discussed in Chapter 2.3 Methodologies in Human Computer Interactions and usability.

5.1.1 Collecting the data

From a previous study performed by Øystein Haugen, we had access to a set of novice-produced models. As a part of a course at the University of Oslo, a collection of students were given a short introductory lecture into BVR, and they were then instructed to model the task given in Appendix I Experimental guidelines/Modelling task from experiment.

The experts were assigned the task of checking and correcting the models in different order, following the experimental procedure given in Appendix I Experimental guidelines/Experimental procedure. The experiment where performed using the experts own office and computer.

The procedure was recorder using Wink (DebugMode), set to capture screenshots every 10 seconds.

The captured data was analysed to identify interesting occurrences, which were later categorisation and prioritising described in Chapter 5.1.3 Categorising and prioritising observations

The novice made models are available in Appendix II Novice Models.

5.1.2 Collection reasoning

We decided to use the novice-made models since they both represent a wide variety of errors and modelling styles, and give a good representation of real world scenarios.

We decided to use Wink, as it is an easy tool to use for screen capture, and all the experts were at least a little bit familiar with it.

Having the experts in their own office and on their own computers minimized the influence of the experimental procedure on the outcome.

Lastly, we used a different work pattern for each the experts since we suspected a learning curve, which can be difficult to separate out from other timing artefacts.

5.1.3 Categorising and prioritising observations

In order to effectively analyse the observations made we have classified the insights into several categories.

First, we have *modelling* observations. Modelling observations are those made on the models themselves. This would include observable problems with the model (other than errors) and any observable difference in the timing on different modelling styles.

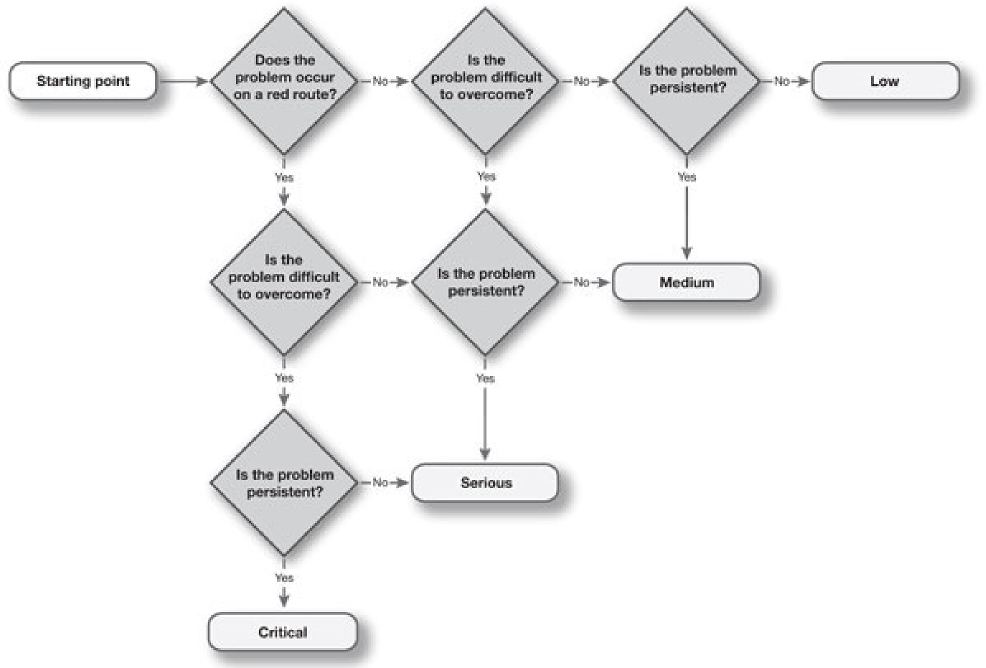
Second is *process* observations. These are observations on what the experts are doing when searching, correcting, rebuilding, and resolving the model.

Third is *tool* observations. This category includes both errors/problems in the current tool and functionality, which seems to be missing.

Fourth and last is *language* observations. These observations are any language specific observations. E.g. the defined look of a Choice or Constraint.

In addition we will grade any problems on a scale from low to critical using a slightly modified version of the definitions proposed by (Travis, 2009), which is designed to prioritize usability improvements (see Figure 33). Where the “red route” is defined as:

“frequent or critical tasks — are the most important tasks that the system needs to support, by definition. For example, if the "on-off" button on your newly designed gadget is hard to operate, all of your users will be affected. Because problems on red routes affect more users, they are more severe.”(Travis, 2009)



Starting point
Does the problem occur on a red route?
No
Is the problem difficult to overcome?
No
Is the problem persistent?
No
Low
Yes
Yes
Yes
Is the problem difficult to overcome?
No
Is the problem persistent?
No
Medium
Yes
Yes
Is the problem persistent?
No
Serious
Yes
Critical

Figure 33 Problem prioritization (Travis, 2009)

While (Travis, 2009) designed the original prioritization definitions to handle usability problems, we have found it to be reliable for other problems as well.

This does require a slight redefinition of the severity levels, as well as a different definition of “persistent” and “difficult to overcome”. We will not consider persistence as something, which occurs throughout the UI, but rather as problems that occurs often. A problem that is difficult to overcome simply means that it is difficult to deal with. An example of a difficult problem to overcome is finding the name of a Choice, when creating a Constraint, since the tool does not allow the modeller to change the view when creating Constraint, or close the Constraint if it’s content has an error(see chapter 5.3.3 Tool mechanics).

Low priority, are problems that cause no particular problems, and we consider them considered mostly cosmetic.

Medium problems are things that taka time, and/or cause frustration, but which do not stop the modeller form successfully completing their task.

Serious problems are the errors that are both persistent and time consuming but not difficult, or problems which are difficult, but which occurs rarely.

Critical problems are those that renders the modeller unable to complete their tasks successfully.

We have labelled the models “P10”–“P15” and we refer to the experts as “A” “M” and “Ø” (e.g. the model “P15” Being corrected by “M” is labelled “P15M”).

5.2 Deviations from procedure

Due to errors in the tool used to record the experiment the M12 and the Resolution of P10Ø recordings are missing, and only the notes taken are available.

As testing progressed, it became apparent that the experts quickly became so familiar with the model, and the resolution, that it is exceedingly difficult to make proper judgements regarding much of the resolution process.

5.3 Observations

Table of Observations

Name		**Type**	**Occurrences**
Real World Applicable Models	Critical	Model	P10A P12A P11M
Naming	Serious	Model	P11 P15
Constraint Validity	Critical	Model	P12
Timing	n/a	Model	All
Insignificant VSpecs	Medium	Model	P11M P12Ø P10M
Choice Duplication and Model Size	n/a	Model	P15
Choice Duplication and Hierarchical constraints	n/a	Model	P11 P13
Real World Applicable models	Critical	Process	P10A P12A P11M
Constraint Validity	Critical	Process	P12
VSpec Duplication	n/a	Process	P11A P11M
Resolution	n/a	Process	P11Ø P14M P13Ø P14Ø P15Ø
Modelling task interpretation	Critical	Process	P15 P12
Adding Multiple Choices	Medium	Process	P11A P11M
Awkward additions	Serious	Tool	All except P14
Copy	Medium	Tool	P10M
Choices optional	Serious	Tool	P10M P11M P11A
Constraint Creation	Critical	Tool	P11A P11M
Auxiliary notation in resolution	Serious	Tool	P13M P14M P10-P15Ø
Name, comment, constraint	Medium	Tool	P11A
Validation Error	Medium	Tool	P13-P15
Size	Critical	Tool	P12A P13A P14A
The visual distance of grouping	Low	Language	P10A P12A

5.3.1 Model

In order to identify areas of improvements to the language, tool, and process, we separate out the impacts and problems caused by the BVR model. Observations made here can serve as pointers to areas, which need extra tool support.

Real World Applicable models

If we want to create a variability model from a given specification, it is important that this model contain all the variations that are specified and no other. A model with these traits is what we mean when we say it is Real World Applicable (RWA).

Even experienced BVR users have difficulty judging whether or not a model is RWA. In Figure 34 we observe an example from our experiment where the corrected model is no longer RWA since it contains a choice, which is not possible according to our specification (i.e. a valid resolution can contain both a parking assistant and a backing sensor).

In P11M, the expert almost misses both the missing parking-assistant/backing-sensor and the naming error.

P10A misses parking-assistant/backing-sensor constraint missing.

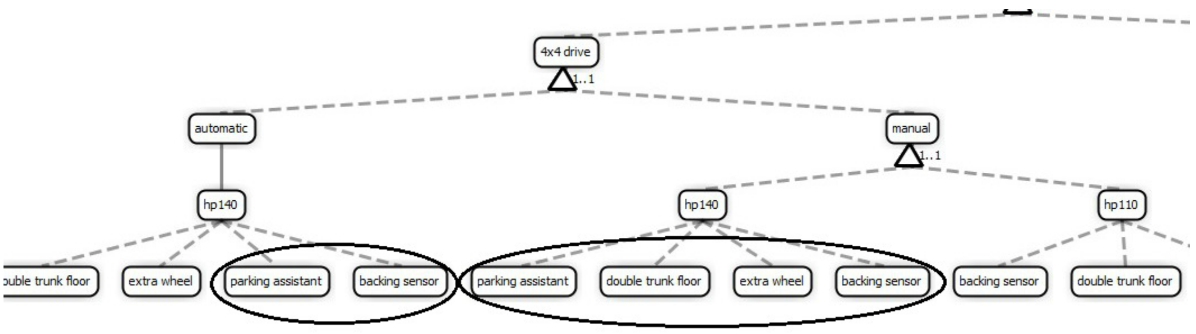
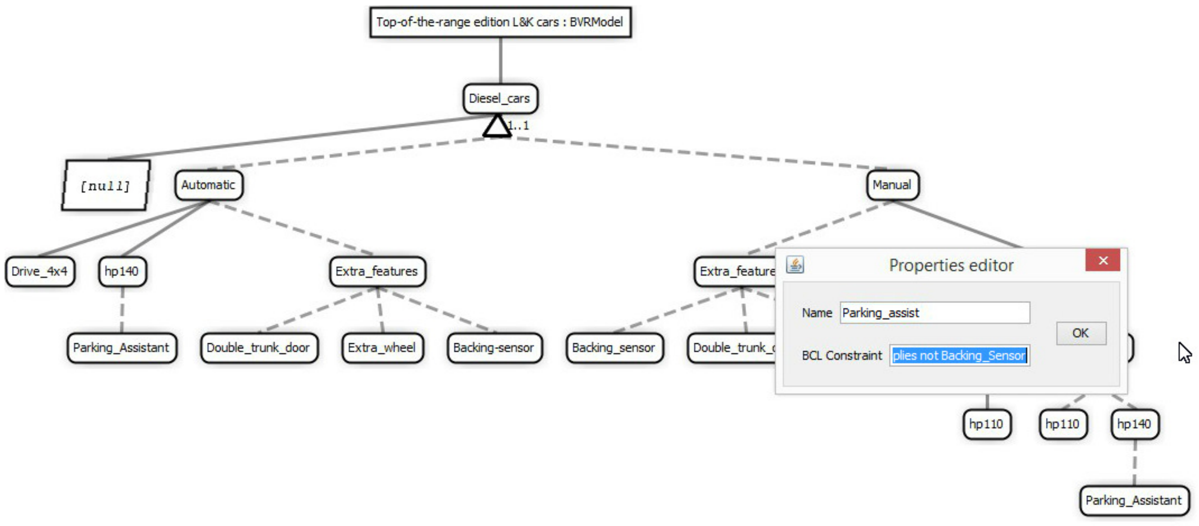


Figure 34 P10A "parking assistant" "backing sensor" constraint missing

Priority: This is a **critical** problem as it is part of a critical task, can be difficult to overcome (i.e. hard to spot, and can require the restructuring of the entire model when it is spotted) and can occur at any time during the build process.

Naming

The problem of names is well known, and outside the scope of this paper, but we would still like to make a couple of observations. In P15, we observe the use of abbreviation in choice names, which in turn degrades the readability of the model. We also observe that there is an inconsistency with the use of capital letters. In Chapter 5.3.3Tool mechanics, Constraint creation, we will observe that there were problems with the naming in relation to the creation of constraints. In particular the problems occurred due to inconsistent use of dash and underscore (-/_). P11MA



Name Parking_assist

BCL Constraint plies not Backing_Sensor

Figure 35 P11M Names causing Constraint problems

In P11Ø and P11A, the expert missed one of the naming errors, which could cause problems in the realization process.

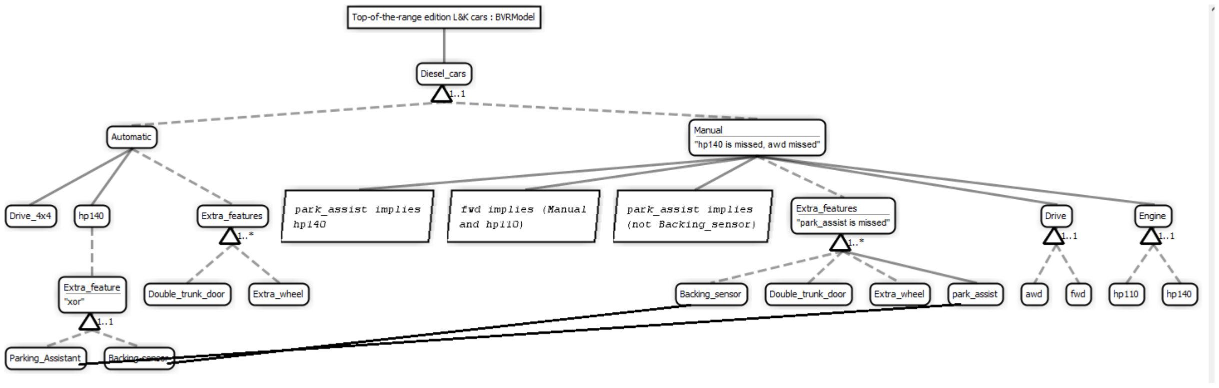


Figure 36 P11A naming cause errors in Realization

Priority: While the problem seems trivial at first, and quite simple to alleviate(but very hard to remove entirely), it is by our definition a **severe** problem as it is a part of the VSpec creation, and can be a problem at any point during the build process but it is easy to fix.

Constraint implications

Dealing with Constraints is a large part of the BVR modelling process, and it can be tricky to deal with, even for experts. We find one interesting example of this in P12. There is a Constraint, which all experts deal with in different manners. The Constrain in question reads

"(Manual and FourByFour) implies (HP110 or HP140)". In its original form, the Constraint does not impart any constraints on the model. There is a possibility that the constraint was meant to imply an "XOR" rule. I.e. that the choice of either engine is only possible when Manual and FourByFour is selected.

One of the experts ignores the Constrain; one expert changes it to "HP110 xor HP140", but only after rendering it useless by enforcing a "1..1" group, and the last expert deletes the Constraint outright after implementing a "1..1" group.

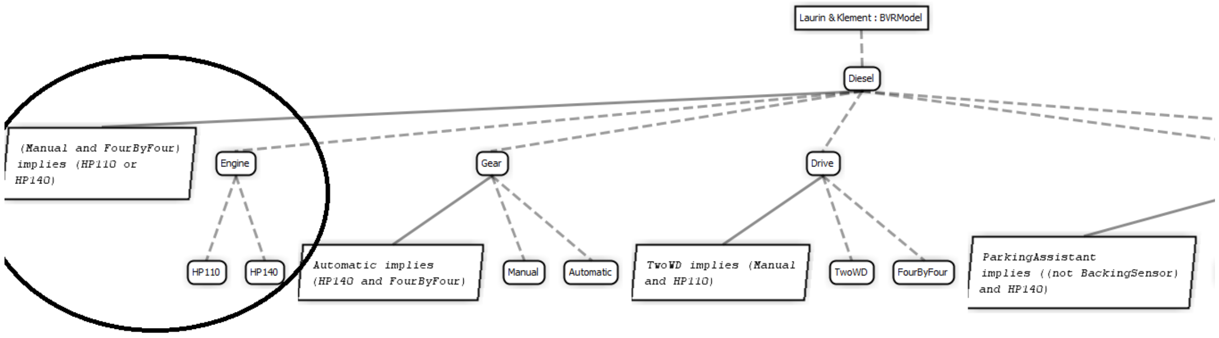

Laurin & Klement : BVRModel
Diesel
(Manual and FourByFour) implies (HP110 or HP140)
Engine
Gear
Drive
HP110
HP140
Automatic implies (HP140 and FourByFour)
Manual
Automatic
TwoWD implies (Manual and HP110)
TwoWD
FourByFour
ParkingAssistant implies ((not BackingSensor) and HP140)

Figure 37 P12 Constraint Validity

Priority: Understanding the implication of Constraints is a significant part of the BVR modelling process and can, as we have just observed, be quite difficult to do. Therefore, while it is often possible to create VSpec models with few or simple constraints, they occur often enough for us to eventually consider this a **critical** problem.

Time spent

One of the metrics we have available after performing the experiment is the timing, which is given in total frames, and the impact of different factors on the experts' ability to perform the task effectively. We will now present the total time spent by the experts on each model, followed by a more detailed breakdown of the numbers. In the detailed breakdown the order column, represent the order in which the expert worked on the model.

Total time spent on model:

Model	M	A	Ø
10	1031	920	3357 + resolution
11	2377	1878	3417
12	2456	1692	1987
13	1566	1042	1562
14	857	1320	1374
15	1057	1950	1775

Time divided into time spent on VSpec, resolution and time spent due to tool error in addition to showing the order in which each expert worked on the model:

Expert	M				A				Ø			
Modell	Order	VSpec	Res	Error	Order	VSpec	Res	Error	Order	VSpec	Res	Error
10	5rd	820	210	-	6th	530	390	-	1st	3357	unknown	-
11	3rd	1400 +687	290	-	5th	1611	267	-	2nd	2610	821	-
12	1st	2456	unknown	-	4th	1050	641	-	3rd	1160	827	-
13	2nd	1020	320	1793	3rd	600	432	-	4th	440	628	-
14	4th	240	617	-	2nd	500	380	425	5th	500	873	-
15	6th	550	250	250	1st	1370	580	-	6th	830 +65	880	-

The timing of the experiment show a few things that were expected. First, the experts spend more time on their first model than their counterparts do. In all but one case (14A 14Ø), the experts use more time than their counterparts do for their second model as well, and in this case the model in question is a model without errors (except the name, see chapter 5.3.2 Modelling task interpretation) and is the one all the experts used the least time on.

Second, the more corrections the experts have to make to the model, the more time it takes. This is clearly visible in P11 where the non-expert seem to have misunderstood either the modelling task, or the language.

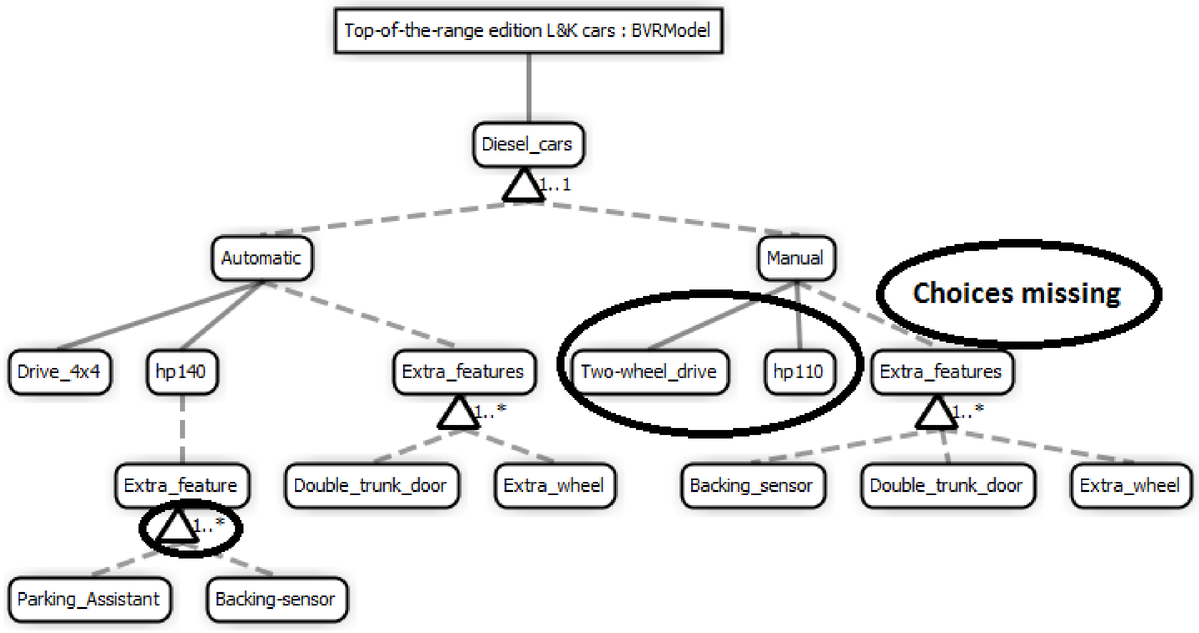


Figure 38 P11 with errors pointed out

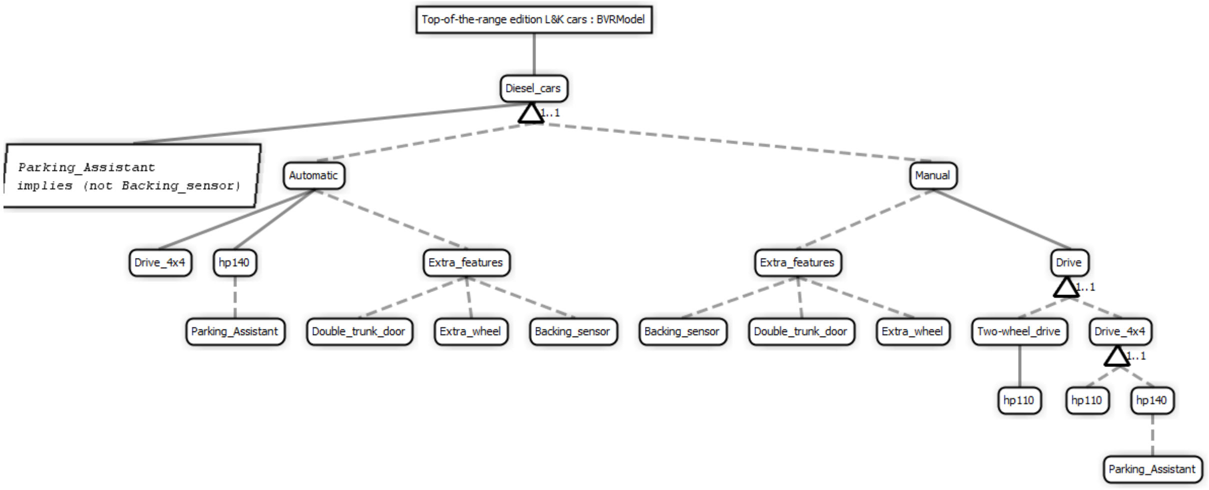


Figure 39 P11 corrected

While the sample size is too small to make any statistical analysis, our data indicates that P11 and P12 were the most time consuming.

While P11 takes time to correct simply because of the amount of errors, P12 is interesting because it shows the problem with Constraints. While the work needed to correct this model is minor (I.e. inserting grouping and correcting the leftmost Constraint), it still takes the experts more time than average to correct. We believe this is because the experts need to keep track of the Constraint logic, and mentally check if they cover all constraints set by the specification. It is also interesting to point out that the resolution of this model is easy if the expert follows a predefined resolution (e.g. "resolve One AWD manual car with parking assistant and double trunk"). During our work with BVR and variability modelling, we have observed that this is not necessarily the case when creating new resolutions on constraint-heavy models (e.g. "make a valid resolution of the model in Figure 40 P12"). It is our belief that an experiment timing the experts making one or more valid resolutions from scratch would show a significant increase in time when resolving Constraint focused models.

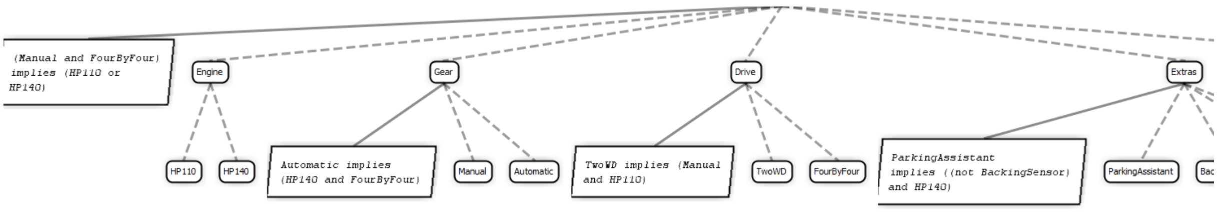

(Manual and FourByFour) implies (HP110 or HP140)
Engine
Gear
Drive
Extras
HP110
HP140
Automatic implies (HP140 and FourByFour)
Manual
Automatic
TwoWD implies (Manual and HP110)
TwoWD
FourByFour
ParkingAssistant implies ((not BackingSensor) and HP140)
ParkingAssistant
Bac

Figure 40 P12

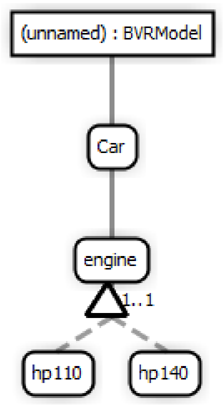
While the dataset is too small to be certain, there seems to be a time benefit to the resolution of more hierarchical structures, P13 in particular.

Priority: n/a.

Insignificant VSpecs

During our work with BVR, we have observed that different modellers have different preferences when it comes to modelling styles. This led us to the use of insignificant choices in BVR, which are choices that in itself does not reflect any change to the product.

If you exchange a Choice (or Classifier) in the model with a Constraint, and the model is still RWA, the choice is insignificant. An example can be seen in Figure 41 and Figure 42.



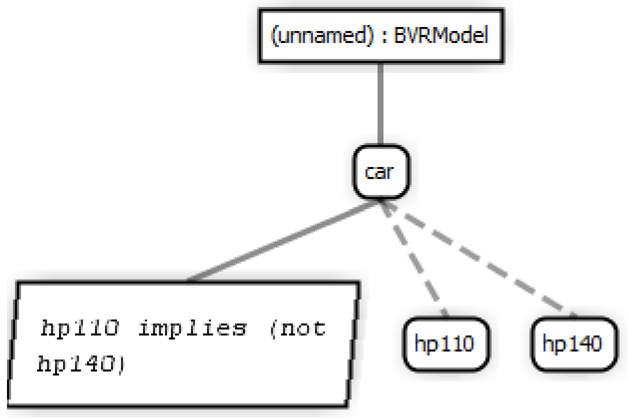


Figure 41 Engine is an insignificant Choice

Figure 42 Insignificant choice exchanged with a Constraint

Insignificant choices occur regardless of whether the focus is on textual or hierarchical constraints. In one version of the use, the parent choice is made optional, while the sub choices are grouped as “1..*”(see Figure 43). In another, the parent, choice is mandatory and the grouping is “0..*”(see Figure 44). A third option is to have the parent choice set to optional and the grouping to “0..*”(see Figure 45). While none of these solutions would affect the realized product, (given that the parent is a true insignificant choice) we believe the first notation is preferable. This notation avoids nonsensical resolutions like “the car has extras, but not really”(i.e. the positive resolution of “Extra_features” in Figure 45 and none of its children.

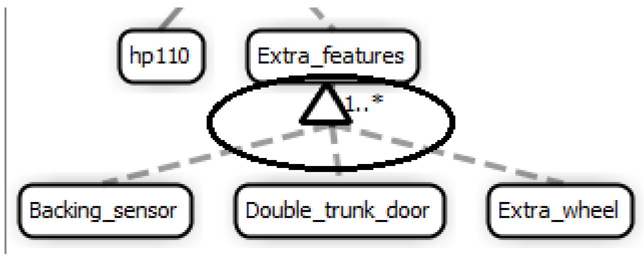


Figure 43 P11 Optional parent, one to many children

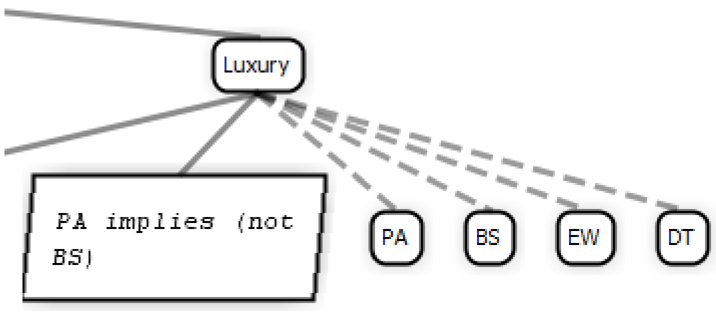


Figure 44 P15 Mandatory parent, zero to many children

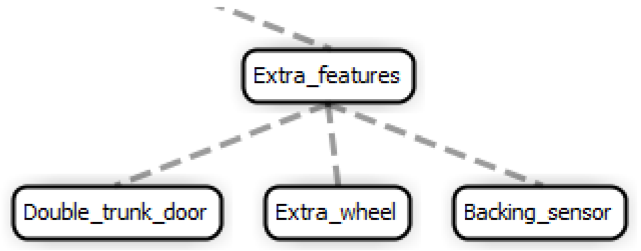


Figure 45 P11 Optional parent, zero to many children

Priority: Dealing with insignificant choices is of **medium** priority. It has only cosmetic impact on the model; it is not something that needs fixing if it is observed. It does however occur on a regular basis.

VSpec duplication and model size

In P15, we observe a complete duplication of the VSpec tree caused by the introduction of the name as a high level Choice. This creates a model that is much larger than strictly necessary (see Figure 46). Which by definition increases the diagrammatic complexity, and thus increases the cognitive load of dealing with the model. This increase in size can also occur in models where the VSpecs are deliberately added to enforce hierarchical constraints, as we will see in Chapter 5.3.1 Model/VSpec duplication and hierarchical constraints

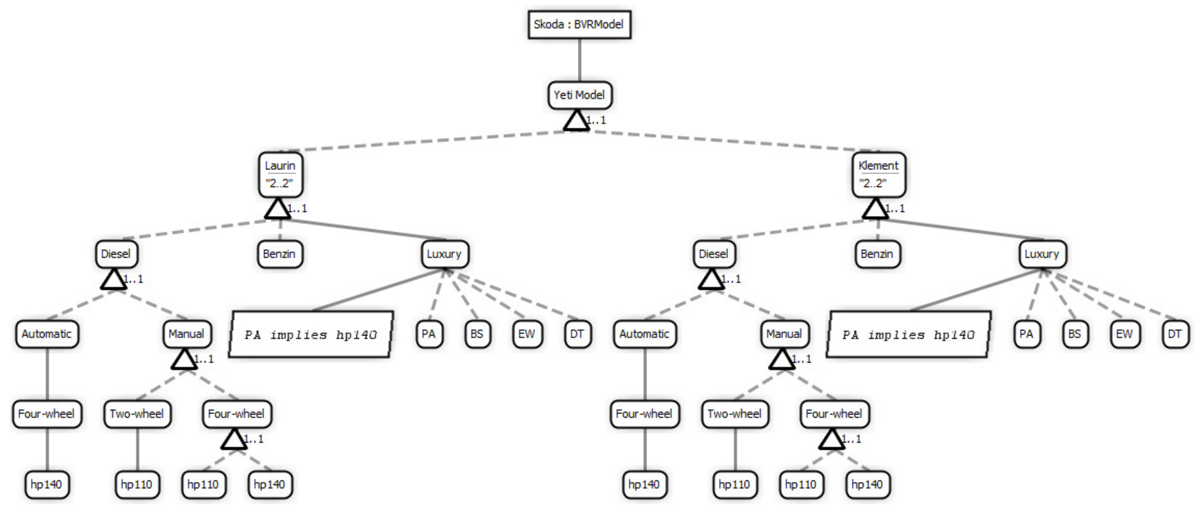


Figure 46 P15 Choice duplication worst case

Priority: n/a

VSpec duplication and hierarchical constraints

In contrast to the chapter 5.3.1 Model/VSpec duplication and model size where we consider the extra VSpecs a problem, we now observe how VSpec duplication can be used to create models with fewer Constraints by implementing hierarchical constraints instead. In P10 and P13 (see Figure 47), we see examples of such models.

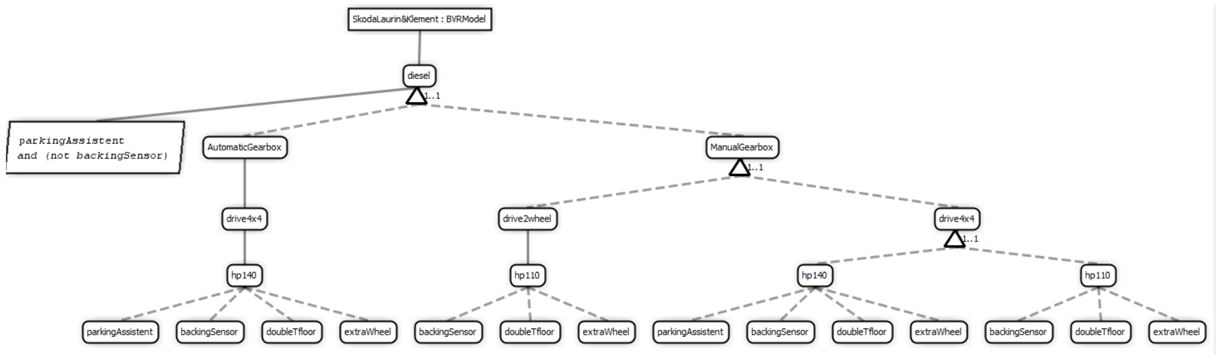


Figure 47 Choice duplication

Priority: n/a

5.3.2 Process

In this chapter, we take a closer look at the processes used by the experts as they work with the models. We hope to use the observations made in this chapter to highlight both tool functionality that would support the experts' way of working while at the same time identifying any situations where the experts would benefit from a defined procedure.

Real World Applicable models

As previously mentioned in Chapter 5.3.1 Model/Real World Applicable models, even experts have problems comparing the BVR model to the specification. From the experiment, we see no evidence of the experts having deliberately created a specific procedure to check if the model is RWA. We later confirmed this observation when talking with the experts.

Priority: The lack of a good procedure is a **Critical** problem.

Constraint validation

When working with Constraints (As we have already described in 5.3.1 Model/Constraint implications) even experts make mistakes. The current process the experts use to validate a Constraint is to examine the logic mentally. We later confirmed this observation by talking with the experts. We believe this procedure works well for short Constraints, but is troublesome when the Constraints grow longer and more complex.

Priority: Just as with the model version of this problem, we prioritize this is as **critical**.

VSpec duplication

In 5.3.1 Model/VSpec Duplication, we observed that models could contain a large number of Duplicated VSpecs. This observation also makes it clear that there can be a great impact from design choices on the complexity of a diagram. Simply moving the name option into a separate Choice creates the model found in Figure 48.

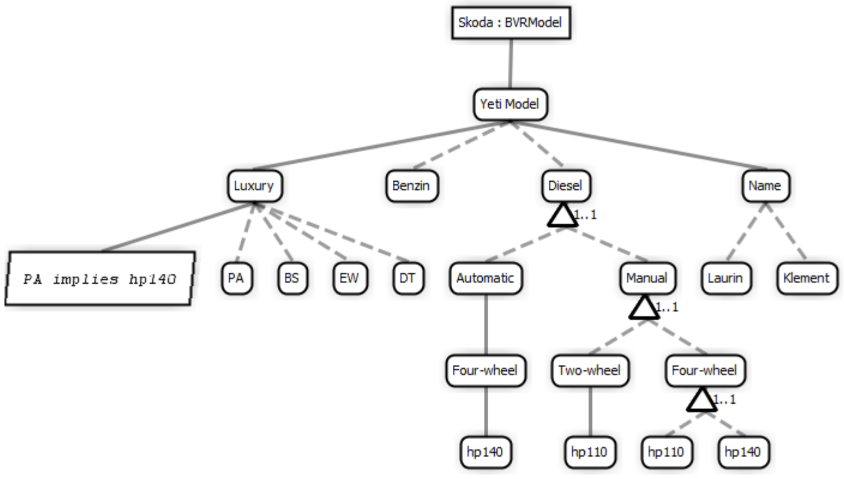


Figure 48 P15 name in separate Choice

Priority: n/a

Resolution

The process the experts use to create resolution models is a good example of a process resulting from the tool. When we look at the resolution method employed by the experts is a top-down approach. In this method the expert automatically populate a fully resolved tree with positive resolution, and sets the topmost unneeded choice to false and works his way iteratively down the tree setting choices to false as he goes.

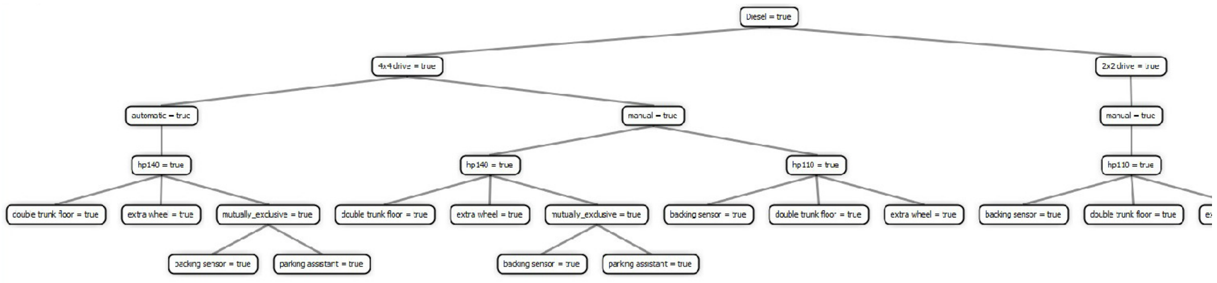


Figure 49 P10 all resolutions present

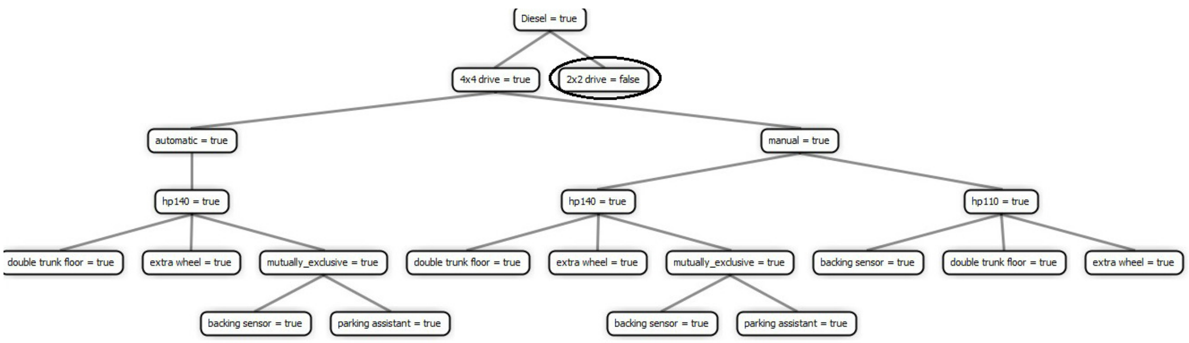


Figure 50 P10 first Choice set to false

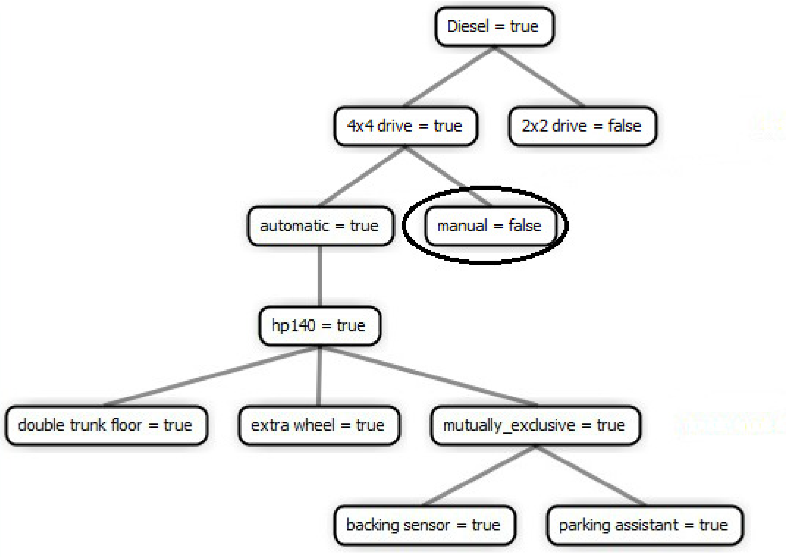


Figure 51 P10 second choice set to false

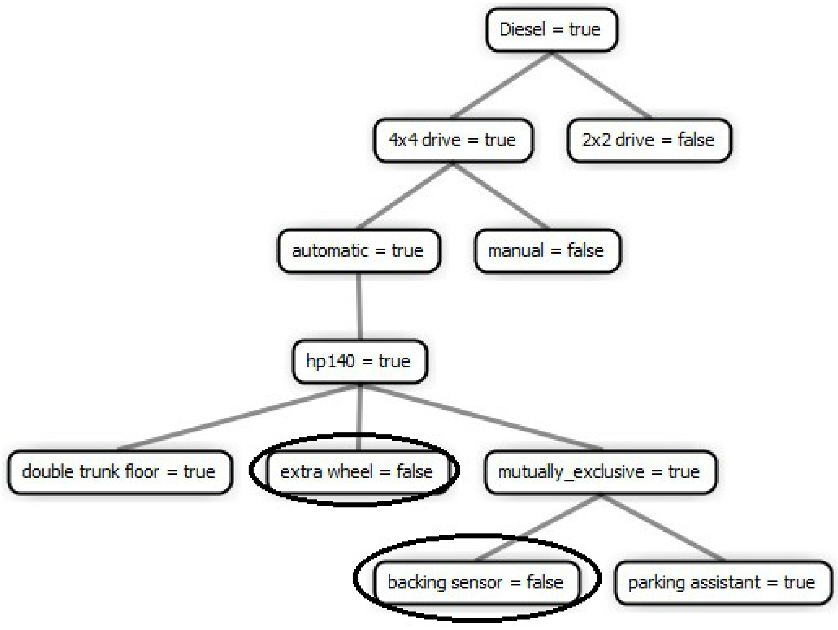


Figure 52 P10 final resolution

This process prevents the creation of resolutions that are incorrect according to the VSpec (e.g. adding more than one manual gearbox). It also seems to be relatively fast, and certainly faster than building the model one VSpec at a time (this is not apparent from this experiment, but is an observation from experience with the tooling and language). In the event that a deep VSpec (e.g. parking assistant) is of importance to the modeller, effective resolution is dependent on a familiarity with the model, which is proportional to the complexity. If, for instance, the modeller wishes to resolve a car with a parking assistant from the running example, he/she would have to search out which of the gearbox and engine combos contained the parking assistant alternative. The result of this is that in a truly large model, finding the correct path to the "deep VSpec" can be very difficult.

While it does not have a direct impact on the RWA of the model, the resolution process is both a large part of the BVR, and a potential place where the modeller can spot RWA problems.

Priority: n/a.

Modelling task interpretation

When we look at the models made by the novices, one of the features that pop out is the frequent inclusion of the model name as a choice, even if it is not a choice according to the specifications. This becomes more interesting when we observe that the experts also disagree on whether or not they should remove the name choice when it is present. Which leads us to a couple of conclusions. The first is that the parsing of a text description without experience or a predefined parsing strategy can lead to misinterpretation of the specifications. The second is that the variability modeller should ideally be both a modelling expert and a domain expert. Both of these observations connects this observation closely to the Real World Applicable Model observations made in chapter 5.3.2 and 5.3.3.

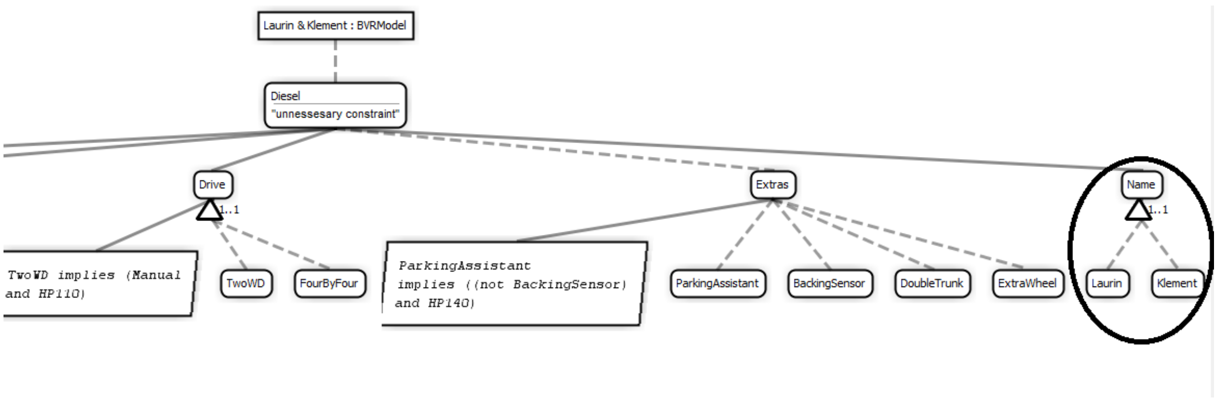


Figure 53 Modelling task interpretation

Priority: Having a process to help recognize valid variations is a **critical** priority.

Adding multiple Choices

One of the actions, which all the experts have to perform is the addition of new VSpecs. In P11 in particular there are several Choices missing from the model. When we look at the process the experts use for adding these Choices, we see something interesting, namely that the experts use two different strategies. In both P11A and P11M, we see the experts adding a several VSpecs before they give names to the Choices (see Figure 54), while in P11Ø we see the expert add one Choice(see Figure 55), name it before adding another one(see Figure 56).

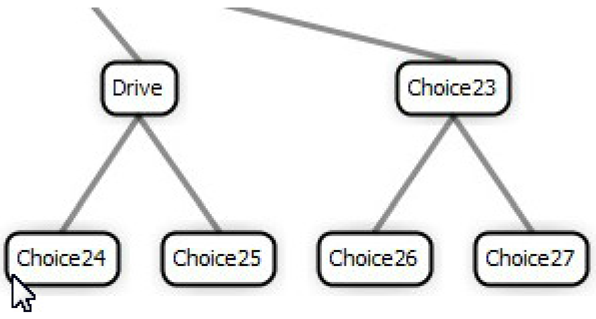


Figure 54 P11A adding Choices

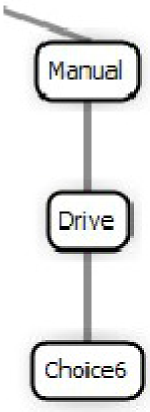


Figure 55P11Ø Adding the first Choice

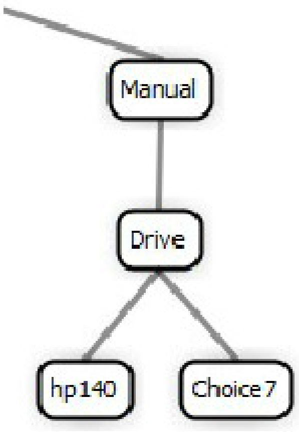


Figure 56 P11Ø adding the next Choice

We believe the tooling would benefit by accommodating both strategies.

Priority: Since only two of the three experts add multiple VSpecs at once, and the current tool seems to work reasonably well with either strategy, this is a **medium** priority problem.

5.3.3 Tool mechanics

In this chapter we describe the tool related problems and uses that where observed during the experiment.

Awkward additions

One of the actions all the experts have to perform when they correct models is the addition of VSpecs. The modeller has to add VSpecs by right clicking and going through a menu as seen in Figure 57. While this technique works, it is not particularly effective, and the BVR tool would benefit from the addition of either a keyboard shortcut or another mechanism, which adds the quick addition of VSpecs.

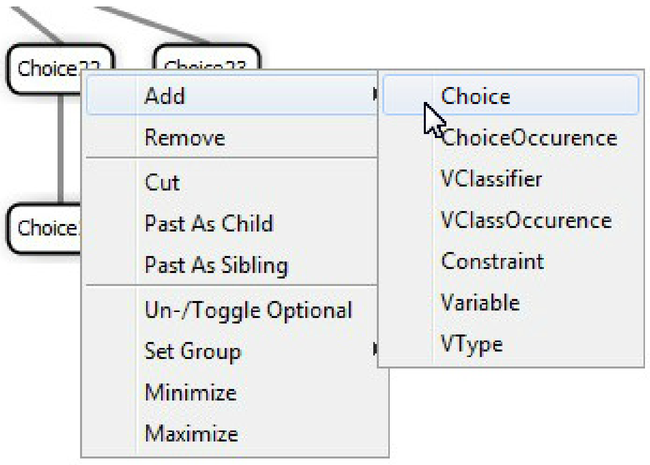


Figure 57 Adding a Choice to P11A

Priority: Adding VSpecs in general and Choices in particular is definitely a part of the "red route", and while the current method is not difficult to use, the modeller adds Choices frequently enough for this problem to earn a **serous** prioritisation.

Copy

In chapter 5.3.1 Model, VSpec duplication and hierarchical constraints we observe that VSpec duplication can be used to implemented hierarchical constraints. Building models using this modelling style, can however cause some of the problems discussed in chapter 5.3.1 Model/Naming, by introducing mode opportunities to create errors.

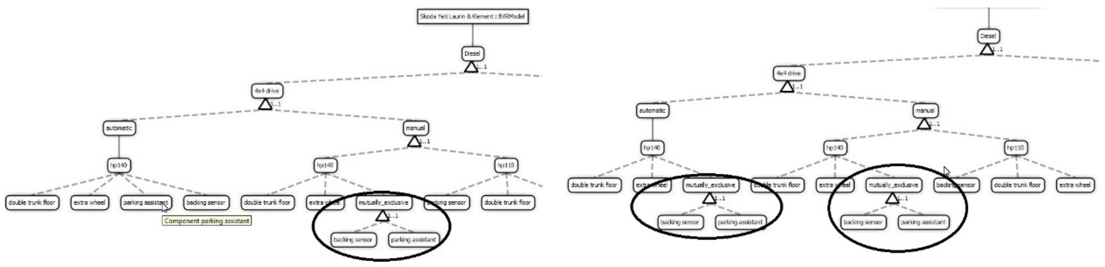


Figure 58 P10 copy structure

Priority: This is a relatively frequent problem, but it is not terribly difficult to overcome, and it is not on the red route so it is a **medium** priority problem.

Choices optional

As a part of the correction process, the experts have to create new VSpecs that the tool creates as mandatory by default. In nearly all cases the experts change this to optional.

This creates unnecessary work, and increases build/correction time.

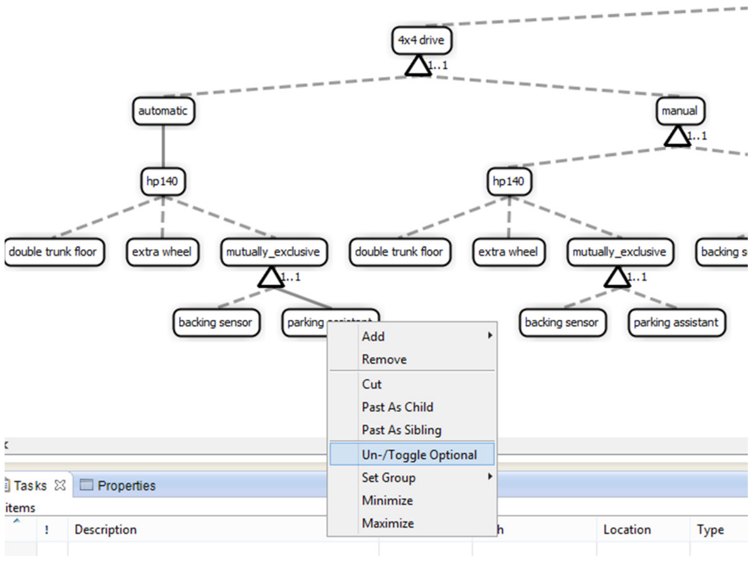


Figure 59 P10 all choices optional

In addition to the time aspect, there is a problem in P11A where the expert forgets to set the optionality of a newly created choice (see Figure 60)

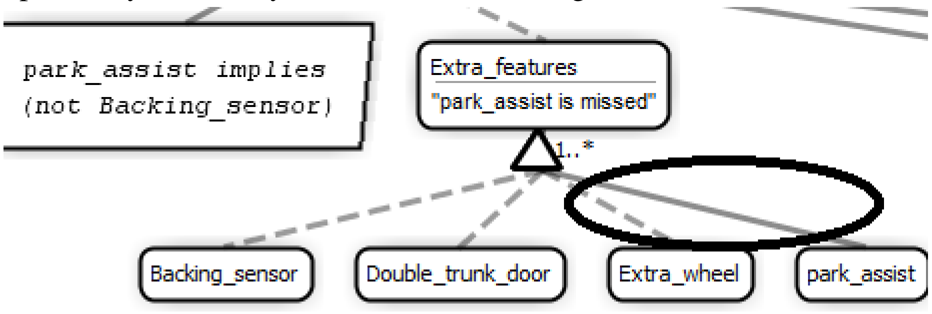


Figure 60 P11A optionality error

Priority: This problem is both along the “red route” and persistent, although it is not difficult to overcome. We can therefor consider it a **serious** problem.

Constraint creation

One of the features of BVR is that it is a strict editor (i.e. it does not allow invalid input), so when the modeller types in a Constraint and presses the OK button or enter, the BVR tool parses the text input and checks its validity. The result of this is that, while you always know that any Constraint in the model is syntactically correct, it will not allow you to enter half-formed Constraints or Constraints pertaining to VSpecs that have not been added to the model. This leads to the following two observed problems observed in P11A and P11M. The first is that the constraints require precise names, which can sometimes be difficult to find, especially considering that the dialog box locks the tool view. The second problem is that the constraints require all Choices meant to be the same to have the same name, spelled in the same way. While the second problem occurs in more or less all programming and modelling languages, it is especially difficult in BVR since it is both machine interpreted and instantiated multiple times without assistance.

Priority: This slows down the creation/correction process as well as making it more difficult to maintain control and certainty over the correctness of the model. This is a **Critical** problem, since it occurs on the “red route”, can be difficult to overcome, and is persistent.

Auxiliary notation in resolution

While Constraints and Grouping are not technically part of the resolution model the current tool does have view modes that show them since they influence which resolutions are valid. Two of the experts use this auxiliary notation, and one of them uses it consistently (P13M, P14M, and P10-P15Ø). In P15, one of the experts uses this tool to spot the error in the VSpec model while resolving, while the other two miss it (see Figure 62 P15 Error missed by experts).

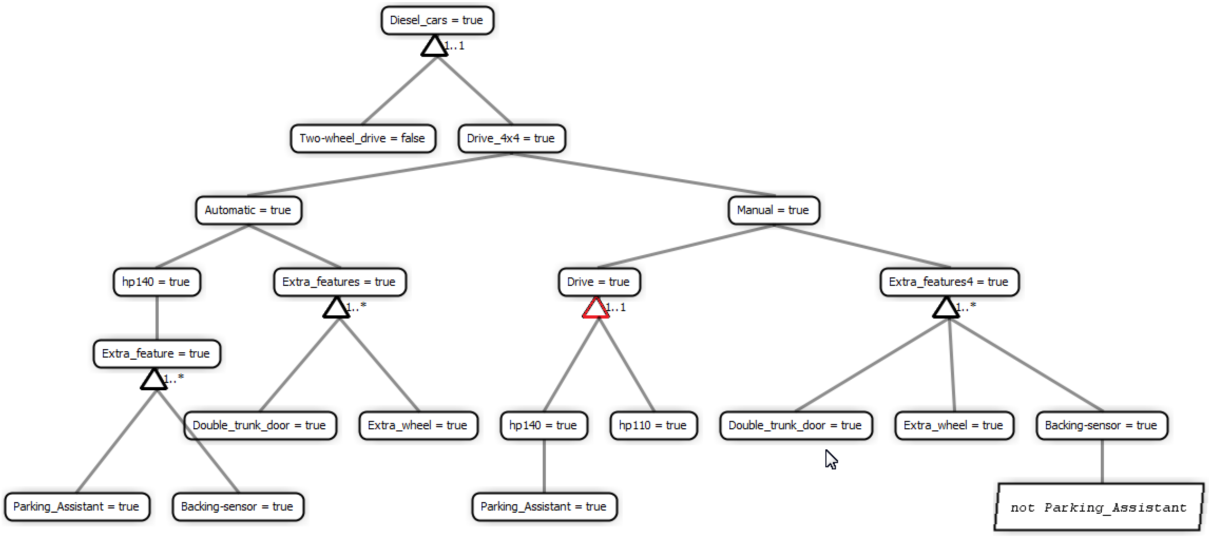


Figure 61 P11Ø Showing VSpec objects in Resolution View

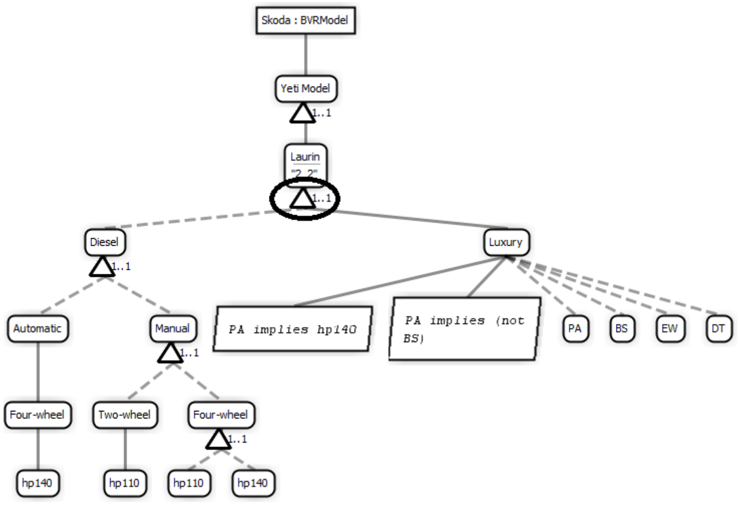


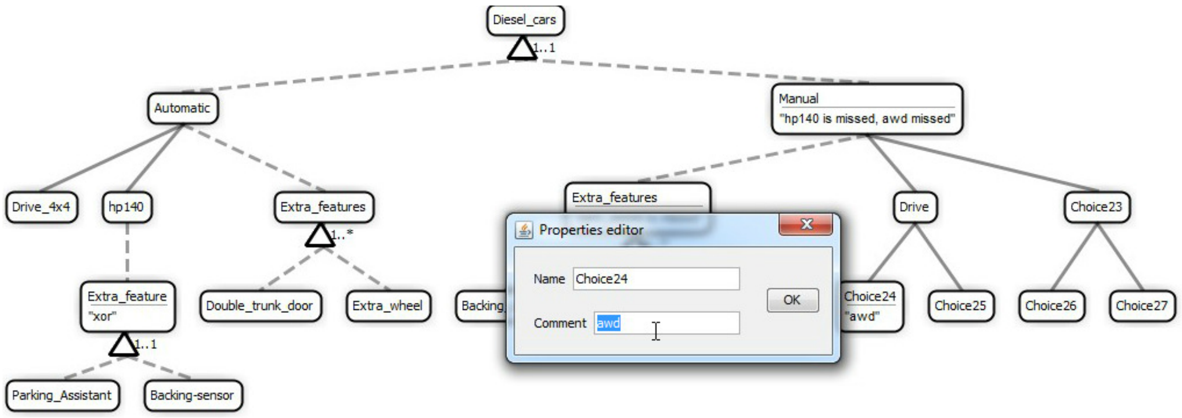
Figure 62 P15 Error missed by experts

There could be a benefit to validating the Constraints in real time when they are shown in the resolution editor, as it could allow experts to identify VSpec errors while they create resolution models in a similar manner as in P15Ø.

Priority: The implementation of this tool would gain a **serious** prioritisation, since Constraint are both difficult to deal with, and are a part of the red route. It does, however not occur to frequently.

Name, comment, and constraint name input

On several occasions (e.g. P11A), we observe the experts typing the constraint in the name field of a constraint. We also observe similar behaviour with the comment field edited in choices, when the expert means to make changes to the name field.



Name: Choice24
Comment: awd

Figure 63 P11A name-comment error

Priority: This problem is part of the “red route”, but it is easy to fix, and is does not occur particularly often, so it gains a **medium** priority.

Validation error

There currently exists an error in the tools validation functionality due to the lack of “target” functionality, which is not yet implemented.

Priority: this is a **medium** priority problem. It is difficult to overcome, but is not a critical task and is only present in some of the models.

Size

The BVR tool does not handle large models well. In particular, it cannot show more than a limited part of the model at once, and it has no indication of where in the model the current view is located. We have observed that experts use time going back and forth to check for completeness and validity. In addition to the models mentioned in the table at the beginning of this chapter, Ø also comments on it in P12.

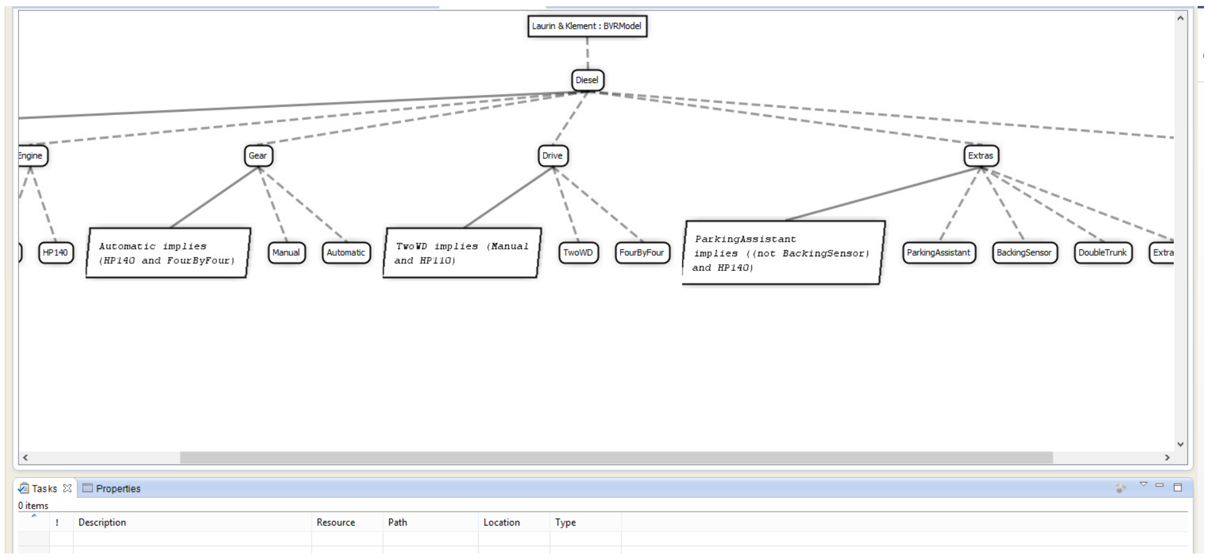


Figure 64 P12, Size problem

Priority: This is a **Critical** problem. Most models are larger than the screen and as such it is both on a red route, and persistent. In addition, it can be difficult and frustrating to overcome.

5.3.4 Language constructs

Finally, we will discuss observation that are directly related to the BVR language constructs.

The visual distance of grouping

As we have discussed in chapter 4 Visual assessment of BVR, the visual distance between the different group multiplicities (i.e. “1..1”, “1..*” “#..#”, and “0..*) is too short to be effective.

While the error in P10A and P12A might be caused by other problems, the errors (see Figure 65 and Figure 66) they could possibly have been avoided if the language expressed the difference between OR and Xor clearer.

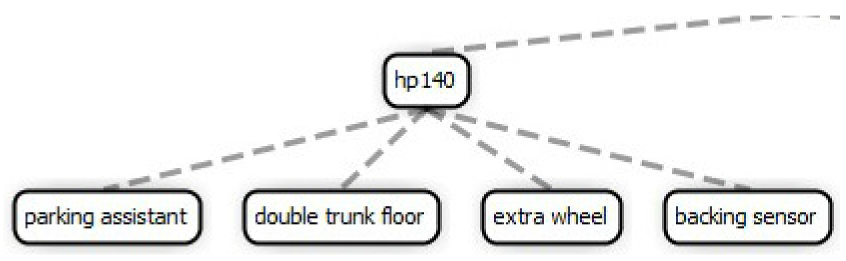


Figure 65 P10A Grouping error

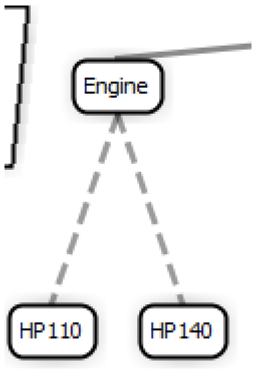


Figure 66 P12A Grouping error

Priority: This is a **low** priority problem. It is not on the “red route”, it is not difficult to overcome, and while the group multiplicity problem is common it is not common to mistake one for the other.

6 Proposed improvements

While we divided the observations into Modelling, Process, Tool Mechanics and Language Constructs, we classify solutions as either tooling, process or language. We do this because the modelling structure flows naturally from the process.

6.1 Tooling

Looking at the problems uncovered during both the experiment and the graphic analysis of BVR, we propose the following improvements to the BVR tool.

6.1.1 Copy

As we have observed in chapter 5.3.3 Tool mechanics/Copy a copy functionality would be useful, when building hierarchical models where similar subtrees are likely to occur. On the other hand, this tool could potentially lead the modeller to create models that contain the problems discussed in Chapter 5.3.1 Model/VSpec duplication and model size.

Implementing a copy function in the program would be trivial using a modified version of the clone functionality described in chapter 3 Pre-experimental work.

6.1.2 Choices optional

As observed in Chapter 5.3.3 Tool mechanics Choices optional, Choices have their isImpliedByParent attribute set as false (optional) considerably more often than true (mandatory) in the given modelling task. It would therefore be advisable to set the standard Choice as optional. It would also be advisable to make the default setting of new Choices available as a preference. Again, implementation of this would be trivial. In addition, it is a serious problem from a usability perspective and we believe the improvement should be implemented as soon as possible.

6.1.3 Constraint creation and validation

The data shows that one of the largest sources of errors and time-waste is the creation and validation of constraints. In both P11M and P10Ø, the experts have problems creating the constraint, and in P11A, the expert misses a problem, creating a model that is no longer RWA. We therefore suggest creating two new functionalities. One, which would make the creation of Constraints easier and one that would improve the ease of validating the Constraint once created.

Constraint creation tool design

In order to help in the Constraint creation process, we have designed a tool, which is particularly well suited to increase the Cognitive Fit for novices. We have chosen to describe the tool's functionality through several iterations.

Instead of creating a dialog, this tool could open a new editor, with a text-field on top of the screen, and the VSpec model shown in the ordinary manner, with the addition of colour and texture to visualize the selected VSpecs. The user should then be able to left-click a VSpec to insert an "implies *the choice clicked on*", and right-click to insert a "(not *the Choice clicked on*)".

The first iteration would look something like this:

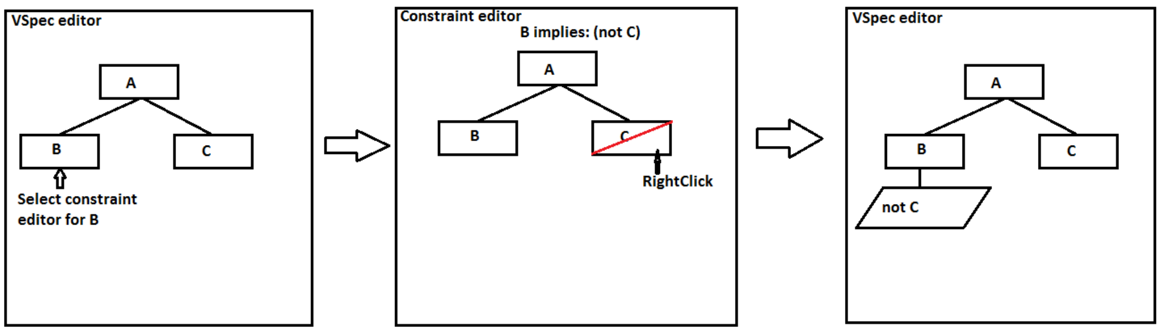


Figure 67 Simple Constraint Creator

The second iteration would account for both the current Constraint validation, which deals with VSpecs through the name, and VSpec Targets (i.e. multiple VSpecs referring to the same object). This is a crucial feature if the tool is to help the validation process. This highlights both unintended consequences and can help find errors in names and the like.

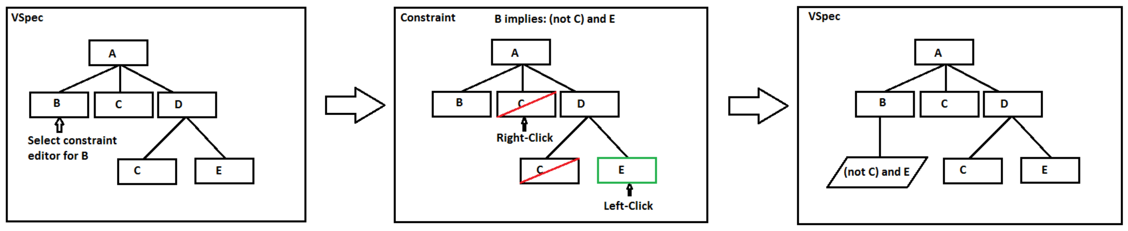


Figure 68 Constraint Creator with targets

The third iteration in the design process implements variables. In this iteration, it would be necessary for the modeller to input a truth assignment related to the variable. The easiest way to do this is a dialog. It would also be useful to note that the dialog should not lock the background, as the modeller might wish to reference other parts of the model.

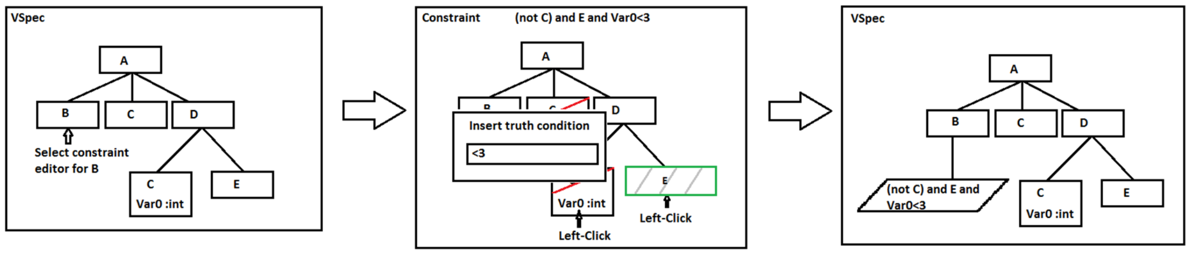


Figure 69 Constraint Creator with Variable editor

The next iteration of the tool creates a problem. In order for the tool to be useful it must be able to handle parentheses, and “not ()”. The implementation of parentheses opens the door for complex expressions, which can be hard to parse and implement graphically. Which leads us to the problem of how to handle a situation like “B implies (not c) and not (E and Var0 <3)” in which E is not in itself a not-statement. We solve this defining anything inside a “not” parenthesis as something shown as negative, even if it is a double negative (i.e. “B implies not (not E)).

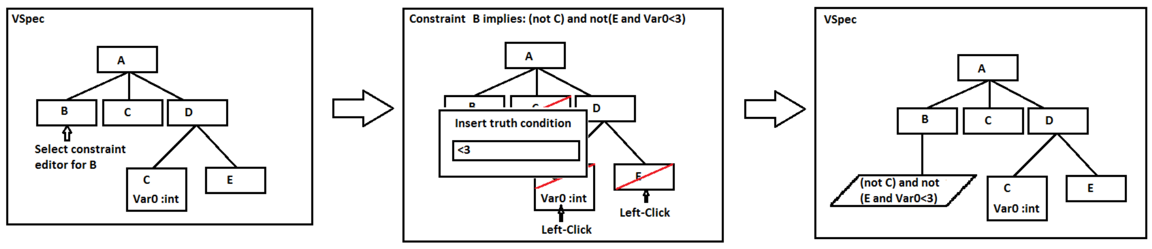

VSpec
A
B
C
D
Select constraint editor for B
C
Var0 :int
E
Constraint B implies: (not C) and not(E and Var0<3)
A
D
Insert truth condition
<3
Var0 :int
E
Left-Click
Left-Click
VSpec
A
B
C
D
(not C) and not (E and Var0<3)
C
Var0 :int
E

Figure 70 Constraint Creator with edited constraint

Finally, there would be a need for and/or/xor. We would choose to implement this functionality with the modifier keys. Holding Ctrl would insert or, while alt would insert xor. This solution makes the constraint building process quick and easy. We have chosen to avoid encoding these logic operations visually.

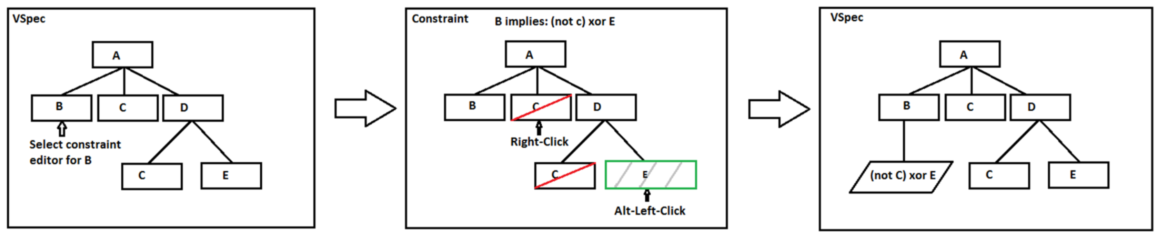


Figure 71 Complete Constraint Creator

We believe this tool would make Constraint creation easy and less error prone than the current solution. We would also suggest integrating it with the tool suggested in Chapter 6.1.10 Size

Tool implementation

In order for this tool to run efficiently, it would be necessary to keep a list or hash-table of all VSpecs except VTypes and Constraints. Since all the names are collected in a hash-map, it is trivial to check if a name is involved in the constraint being created. Deciding if the VSpec should be shown as positive or negative on the other hand, require us to check the rest of the expression to see if it is inside a "not" parenthesis. This tool leads to an additive process, in which the correct Constraint syntax is simple to implement.

The implementation of this tool should be trivial, since the new Constraint editor's UI could utilize the existing code used by the VSpec editor with mostly minor modifications, and the logic can be implemented as described.

Other Constraint creator tool improvements

In addition, we believe it would be beneficial to keep the current text editor allowing for quick creation of constraints. To alleviate some of the problems already mentioned in this text editor could be expanded to include:

1. Some form of auto-completion of names.

2. The ability to edit VSpecs without closing the window

3. Visual cues to which VSpecs are impacted, similar to the graphic interface. This could be particularly useful if we integrated it with the map suggested in Chapter 6.1.10 Size, by colour/texture encoding the affected VSpecs in the map.

Constraint validator tool

The other problem we have observed when working with constraints is the process of RWA validation. While it is impossible to check if a Constraint conforms to the real word, we could make it easier for the modeller to keep track of the VSpecs involved in a Constraint, and the Constraint's implications if the modeller could select a Constraint, and have its impacted VSpecs highlighted.

6.1.4 Names

While the debate on the best practice of naming schemes and the use of Pascal case, camel case or uppercase is fascinating, it is outside of the scope of this thesis. It is, however obvious that some tooling support is required if a better workflow is to be attained. We would suggest the implementation of an autocomplete functionality similar to the one used in LINUX or Eclipse(Eclipse-Foundation). Alternatively, it would be possible to use the edit distance between names, which would inform the user if a name were within a short distance of a previously used name. This functionality, which again would be trivial to implement, would in our opinion, greatly decrease the odds of a situation like the one in P11.

6.1.5 Name, Constraint, and Comment

In 5.3.3 Name, comment, and constraint name input, we observed experts using the wrong input field on several occasions. Since it is safe to assume that the experts understand which text field they are supposed to use, we can conclude that the position of the input fields located in an unintuitive location. While this does not create large problems for the modeller, it does wastes time and it breaks the workflow, which in turn can distract the modeller from their task. A possible solution is to move the descriptive fields to a properties dialog accessible through the dropdown menu. Another is to create a "properties" button to the current properties dialog, and move the less used fields there.

6.1.6 Resolving mandatory Choices

One of the ways of decreasing the cognitive load of resolving BVR models is to decrease the number of apparent choices that can be made. From the BVR metamodel we find that negative resolutions do not have any child nodes. There is therefore no need to show "= true" in the name of mandatory choices, since they can only be resolved as false if they do not exist (see Figure 72 Resolving mandatory Choices).

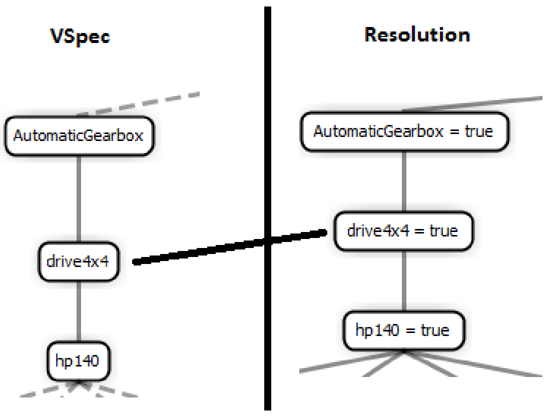


Figure 72 Resolving mandatory Choices

6.1.7 Alternative resolution View

We have observed in 5.3.2 Process, Resolution, a distinct preference for the top-down approach for resolving models.

This resolution technique opens the door for a view mode, which hides false choice resolutions completely from the model. A note made by one of the experts corroborates this idea. Using this view builds in another layer of Complexity Management. Logic dictates that this should be coupled with the grouping and only kick in when a grouping choice is completed, and quite possibly only when the choice is either xor, or a leaf. Applying this view to the P12M resolution shown in Figure 73, would display it as shown in Figure 74

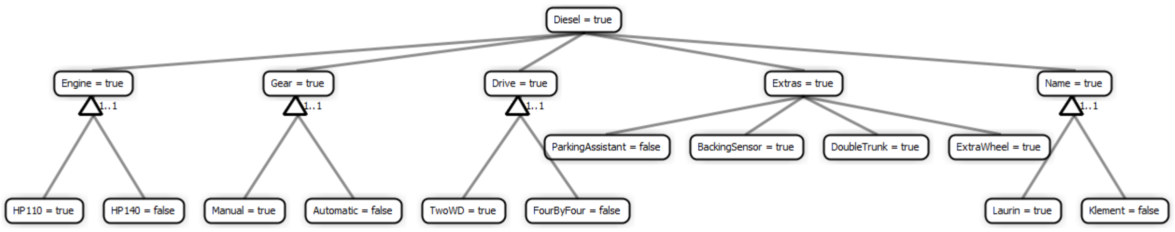


Figure 73 P12M resolved

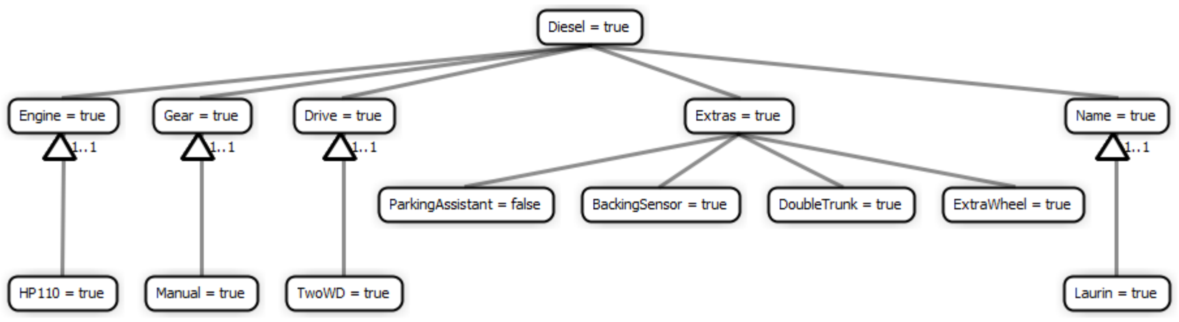


Figure 74 P12M resolved and shown with the alternative resolution view

6.1.8 Resolving with Constraints

Since resolutions can be used to find RWA problems with the model, we believe, as we discussed in Chapter 5.3.3 Tool mechanics/Auxiliary notation in resolution, that an auxiliary notation, which shows Constraint errors in real time, could be beneficial. Showing Constraints which are not satisfied in red, with a different texture would make errors easy to spot. Barring extremely complex Constraints, this should not cause any problems during runtime, since the time to check a constraint is O(m) where m is the number of clauses as long as there is a hash-map of the Resolutions.

6.1.9 Checking the applicability

As we have observed in 5.3.1 Model/Real World Applicable models, even experts working on a relatively small model have problems spotting errors. While we hope that the improvements suggested in this thesis will minimize the risk of making these errors, we also see the need to create either a tool or a method, which would help the experts find such errors. During our work with this project, we have found four possible ways check the RWA of a model. These are Comparative Tool, Impossible Resolution, Visual Inspection, and Conflicting Constraints. They will be discussed in detail in the following sub-section.

Comparative tool

The first is the “Comparative tool”. In this method, the domain-expert create a second model, preferably in a different style (e.g. flat/hierarchical), while maintaining the names from the previous model. The modeller will then proceed to use the “Generate All Products” functionality of the BVR tool, which at the current time only works on models with Choices and Constraints. Using this method requires that the modeller specify which VSpecs represent significant choices (e.g. Figure 75 and Figure 76). VSpecs must have unique names if they represent different things (i.e. similar to the way Constraints work).

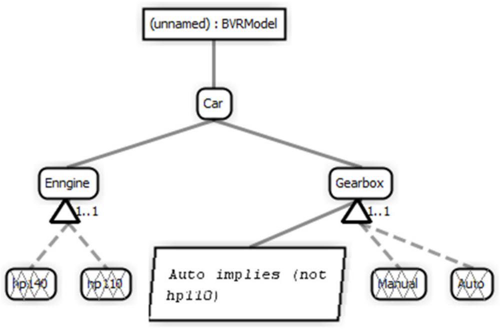


Figure 75 Flat structure Example

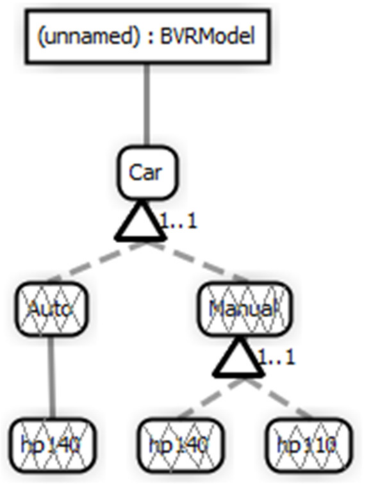


Figure 76 Hierarchical structure example

Once the modeller selects the significant VSpecs, the tool can run the "Generate All Products" and compare the resolution of significant choices for the created resolution models.

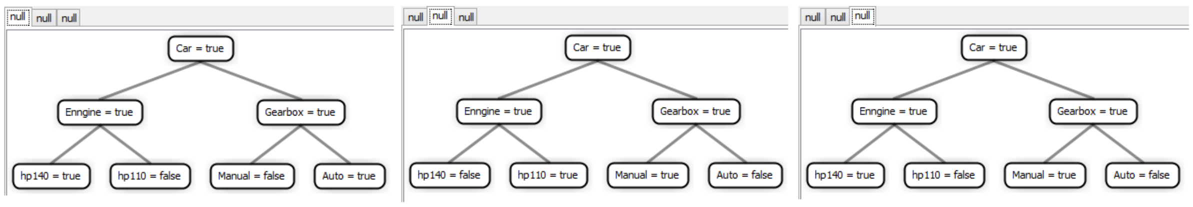


Figure 77 All Resolutions flat structure

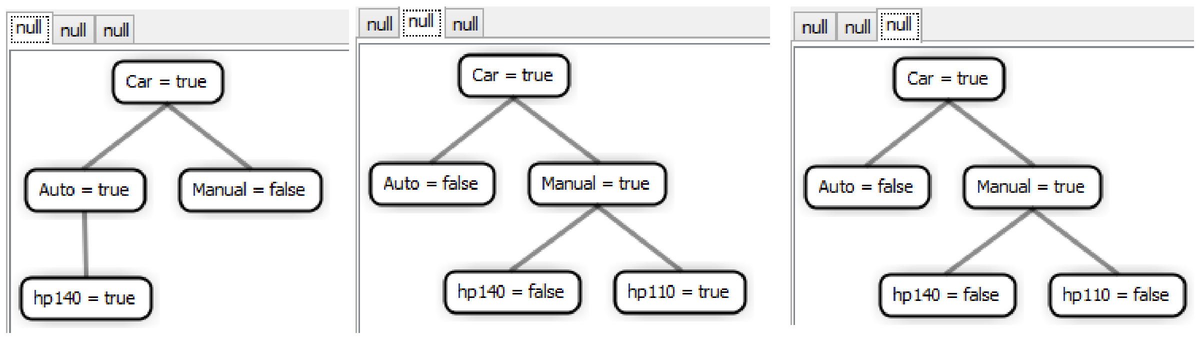


Figure 78 All resolutions hierarchical Structure

The advantage of this method is that the chance that the modeller implements the same error twice is inverse proportional to the size of the model. This means that the largest models, which are the hardest the check for RWA problems have the most to gain from this approach. The downside to this method is that it requires the creation and comparison of several (easily several thousands) models, which is not a quick process. A naïve solution is highly inefficient. E.g. for every significant Resolution in a Resolution model, search through every Resolution in each Resolution model for the opposite structure. Do this for every Resolution model of the first model. This gives us a complexity of $O(mn * (p^2)$ where n is the size of the model, m is the number of significant choices and p is the number of Resolutions created by one of the models (we assume that both VSpec models creates the same number of Resolution models).

One solution, which is far more efficient, would be to make a new version of the "create all products" tool, which incorporates a number of changes.

1. The new tool should only create possible products for significant choices.
2. The new "resolve all"-function would need to work with all the built in VSpec-constructs. Which would also require the modeller to input values to any Variables, since the default values could be insufficient to find all problems.
3. It would not need to build the Resolutions in the BVR model; only fill in the value of the significant choices in a suitable data-structure.

After the creation process is finished, the tool compares the data to see it the collection of Resolutions for significant VSpecs are equal.

Although the solution described above is much faster that the naïve method, it is still slow, and can be effectively non-functional for large variability models.

Impossible resolutions

The second is the "Impossible Resolutions method". Here a domain-expert would either create resolutions that should be impossible, or a tool would automatically create resolutions and the domain expert would go through them to check for impossibility. Neither of these methods require new tooling. There is, however a significant difference in the work required versus the confidence that the model is RWA. If we choose the auto-resolve solution, the expert will have to check a large number of models (88 for the relatively small example seen in Figure 79) for impossible resolutions, but the experts will encounter all existing impossible resolutions (although he/she might not notice them).

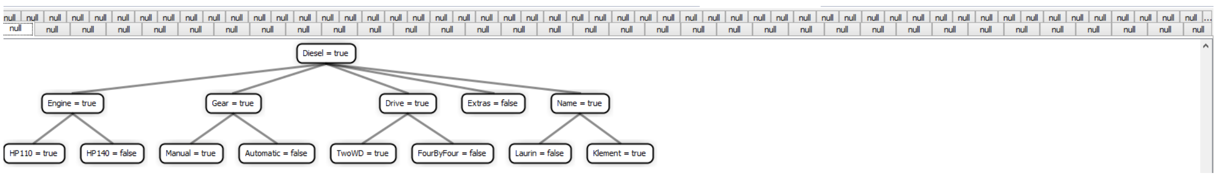


Figure 79 P12 All Products

While this method does not require any new tooling, it does require the expansion of the "create All Products" functionality, which, at this time only works with Choices.

The second alternative of the Impossible Resolutions method requires the modeller to create and validate Resolution Models that should be impossible. This version has a significant advantage over the auto-generated method. It does not require the modeller to look through the same number of resolutions (which could easily reach the several thousands). It does not guarantee that the expert finds all the RWA related errors, but the confidence in the model grows as the domain expert creates and invalidates more resolutions.

We would recommend the second alternative, if someone wanted to implement the Impossible Resolution method.

Visual inspection

The third solution is the “Visual Inspection tool”. This tool would be a visual editor similar to the current VSpec editor, which would in turn highlight different aspects of the model. This type of tool could also be implemented in two different ways.

The first alternative to the Visual Inspection Tool is a VSpec focused one. This tool would iterate through the model and through graphic highlighting show all constraints (group multiplicities, multiplicity intervals, and Constraints) on the VSpec.

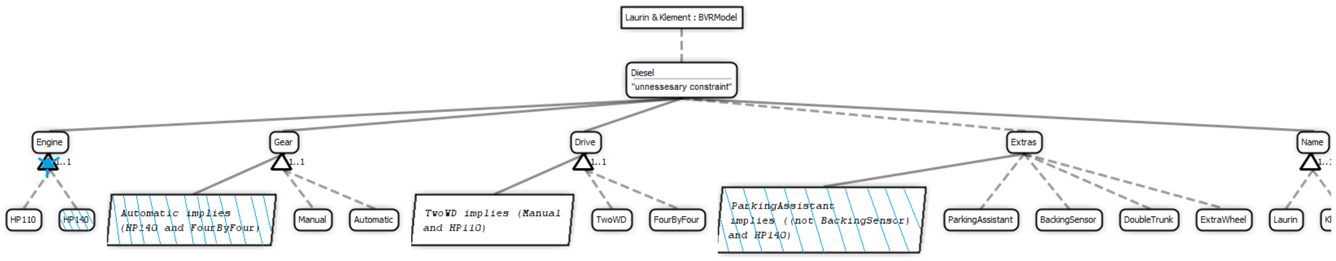


Figure 80 VSpec focused visual inspection tool focused on HP140

The second alternative is a constraint focused tool, which would graphically highlight one constraint (Constraints, group multiplicities, and multiplicity intervals), and the VSpecs it affects. In Figure 81, we see an example of this tool when it is focused on a group multiplicity constraint. In Figure 82 we see the same tool when it is focused on the Constraint “Automatic implies (HP140 and FourByFour).

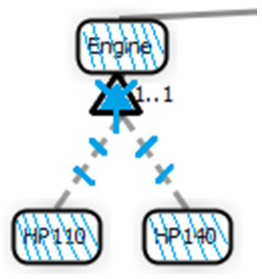


Figure 81 Visual inspection of group multiplicity with constraint focused tool

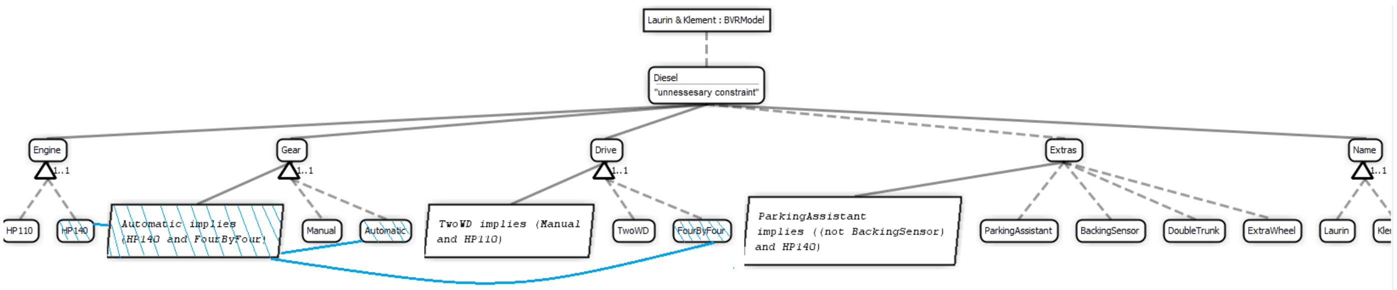


Figure 82 Focusing on Constraint with constraint focused tool

Conflicting constraints

The fourth solution is the "Conflicting Constraints method". This entails the creation of a separate model containing Constraints that should render the model invalid. The tool would then go through the VSpec model, validating the model with each of the Conflicting Constraints. We believe this solution, coupled with the Constraint Creator, would be les labour intensive than both the Comparative and the Impossible Resolution methods, while being more exact then the Visual method. If we take the model in Figure 75 Flat structure Example, and create a Conflicting Constraints model, it would look something like the model in Figure 83.

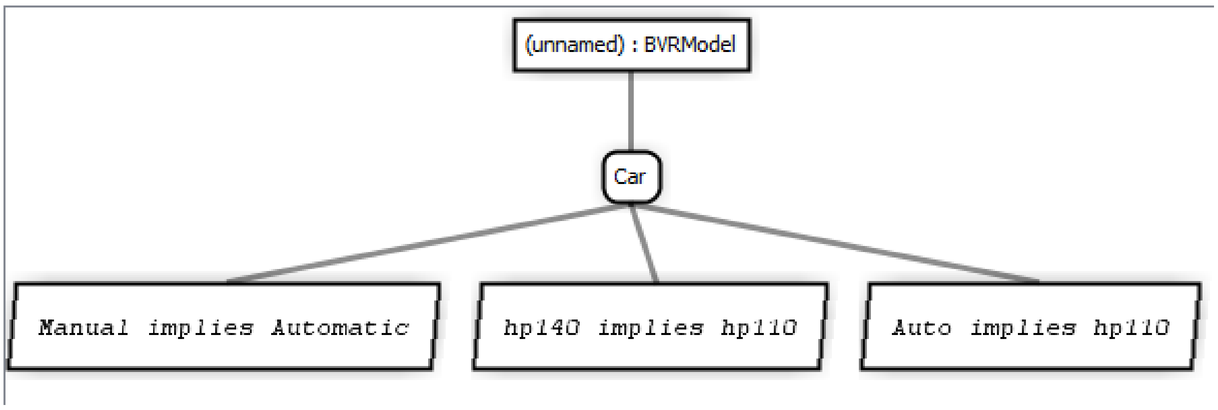


Figure 83 Conflicting Constraints for Flat Structure example

All of these techniques have the same problem, namely that none of them guarantees error free models. The advantage of the second and fourth technique is the ability to increase iteratively the certainty of the models completeness.

6.1.10 Size

As we have previously discussed in Chapter 5.3.3 Tool mechanics/Size, the BVR-tool does not deal well with large models. We believe it would benefit from several improvements, including the ability to zoom in and out, a map for quick navigation and a search functionality.

Zoom

This improvement is quite self-explanatory. We simply believe the tool would benefit from the ability to zoom in and out of the model similar ordinary office software. We would suggest implementing both a hotkey solution (e.g. the + and – buttons) and a mouse scroll-wheel solution (CTRL-scroll).

Map

The second improvement we would suggest is a mini-map, like the one shown in Figure 84

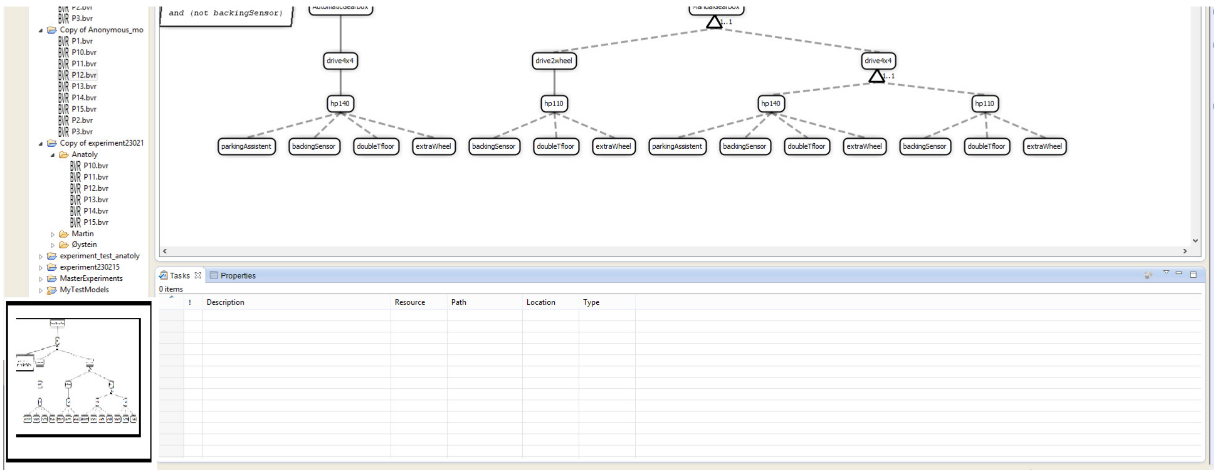


Figure 84 Mini-map

Search

The last improvement we would suggest for dealing with the size problem is the implementation of a search function. We believe this functionality would be the most useful if it were implemented in the following manner:

- It should implement standard search terms and functions e.g. "*" (any string) or "?" (any character).

- It should be easy to iterate the results.

- If it is implemented with the above-mentioned mini-map, it should use colour/texture to highlight the results. Regardless of map implementation it should highlight the results in the model.

6.1.11 Search and auto-resolve

In chapter 6.1.9 Checking the applicability, we discuss the possibility of checking the applicability of a model through Resolutions. In Chapter 5.3.2 Process/Resolution, we mentioned that the process of resolving for a "deep" VSpec can be difficult, and exceedingly so for large models. The result of these two facts is a need for a tool, which lets the modeller search for a Resolution and shows the path from the selected start-node to all search results. One possible implementation can be seen in Figure 85.

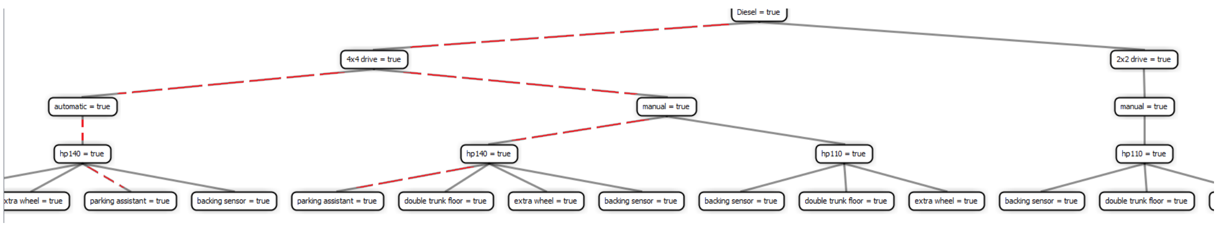


Figure 85 Showing path to Parking_assistant

The implementation of this functionality can be done through a simple exhaustive depth-first search(see Figure 86).

```
searchResolution(resolution){
    if(resolution.name == searchName) {
        resoluion.markAsRoute
        return true
    }
    else{
        boolean route = false
        for (child in Resoluiton.children) {
            if(searchResotluion(child)){
                resolution.Markas route
                 route = true;
            }
        }
        if(route == true ) return true;
    }
return false
}
```

Figure 86 Search and mark route to Resolution

6.1.12 Cognitive integration

As we discussed in Chapter 4 Visual assessment of BVR, we need to create a view that helps the user in creating a coherent cognitive map of the system. We believe we can make two valuable additions to the current tool.

VSpec Type auxiliary notation

The first implements the Conceptual Integration principle, by implementing an auxiliary notation where the modeller can selectively show the contents of an Occurrence (the VType) in the VSpec model. This addition is easy to make, simply iterate the VType-tree, and show the contents as child nodes of the Occurrence. One interesting side effect of this implementation is the question of allowing modellers to make changes to the type in the VSpec model. On the one hand, it would make it easier for the modeller to make changes, on the other, it would quickly seem inconsistent to the novice. This leads us to believe we could allow it, but only through a property setting so the modeller has to explicitly turn the functionality on. In order to comply with the Complexity Management principle, this view should also be optional and localized to selected Occurrences. We also believe that the tool should use the free visual variables (colour, texture, and brightness) to distinguish this subtree from other parts of the VSpec model.

Type navigation

In 6.2.2Types and Type shadowing, we discuss the problematic way in which BVR deals with VType shadowing. Depending on the resolution to this problem, we have two different solutions.

If the solution is to force unique names, we could create a type map as a simple list. This list should be easily accessible, and both clickable (i.e. clicking on a type opens the type editor for the selected type), and searchable.

The other solution is to enforce Occurrence VType assignment in accordance with the natural hierarchy created by types.

6.1.13 Better additions

In Chapter 5.3.3 Tool mechanics/Awkward additions, we observed an inefficiency in the way we add VSpecs. One possible solution to this is to add CTRL-click, as a shortcut to add a new Choice. Since Choices are, in our experience, the most common VSpec this is the most important one to implement. Adding similar hotkeys for the other VSpecs would be trivial.

6.2 Language improvements

6.2.1 Semiotic clarity and perceptual discriminability

As discussed in “4 Visual assessment of BVR”, the current choice of graphic symbols is not ideal. We could improve both the Semiotic Clarity and Perceptual Discriminability by introducing new symbols. Figure 87 Alternative set of Shapes, shows one such set of shapes. This new set introduces new shapes for most of the semantic construct (except VSpec variables and VTypes which are textual and Choice/ChoiceResolution which stay the same), and employs several shape families, which further increases the visual distance. This selection even has the benefit of adding two mnemonic devices for added Semantic Transparency. These are that VSpecs without children from types have downward facing points, and Classifiers look like crosses. As we discussed in Chapter 4 Visual assessment of BVR, there are limits to the number of graphical objects which modeller, and especially novices, can handle. We believe that the coding we have suggested groups the Groups together well enough visually, to allow novices to remember what the group means, while the Dual Coding allows them to discern their meaning. This solution eases the cognitive load for experts when discerning the multiplicity of the Group.

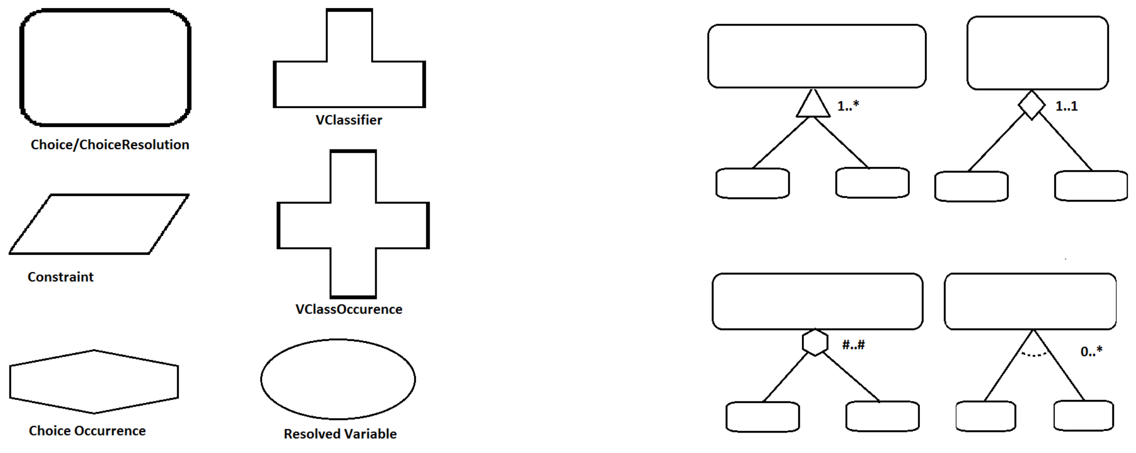


Figure 87 Alternative set of Shapes

In addition to these changes, one might argue that including colour and texture could be beneficial. It is important to note that colour should not be used to encode information without texture, both to avoid colour-blind discrimination and in situation when colour is not available (e.g. black and white printing). On the one hand, colour/texture is quickly perceived, and would therefore help distinguishing, for instance, a VClassifier from a VClassOccurrence; on the other, it has limited capacity, and we would argue that it would be of greaten use in highlight problems(e.g. the red grouping error).

6.2.2 Types and Type shadowing

In its current state BVR does not deal with type shadowing and allows types with the same name. While it is useful to give Occurrences VTypes regardless what VSpec the VType belong to, this can also cause some problems. Specifically, it can cause unpredictable behaviour when a type contains an occurrence of a type that is declared two different places. If, for example, we have the following scenario: The Root defines two types, Type1 and SomeType as in Figure 89 Shadow problem root. Here we define SomeType as shown in Figure 88 Shadow Problem SomeType in Root.

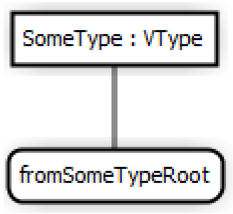


Figure 88 Shadow Problem SomeType in Root

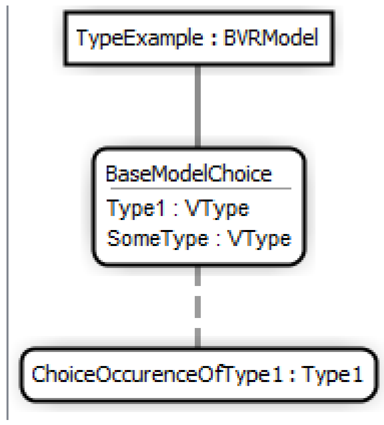


Figure 89 Shadow problem root

Type1 then defines Type2 and a ChoiceOccurrence of SomeType, as in Figure 90 Shadow Problem Type 1 with one Occurrence.

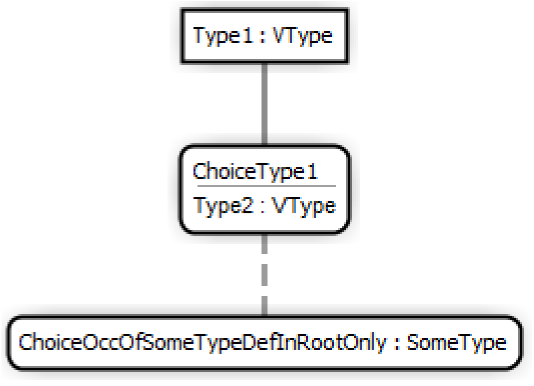


Figure 90 Shadow Problem Type 1 with one Occurrence

We continue by defining Type2 as shown in Figure 91 Shadow Problem Type2, with the new SomeType defined as shown in Figure 92 Shadow Problem SomeType defined in Type2.

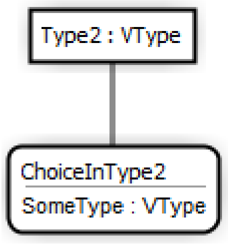


Figure 91 Shadow Problem Type2

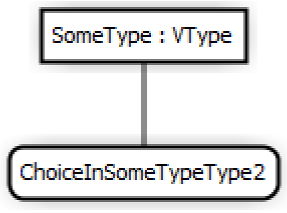
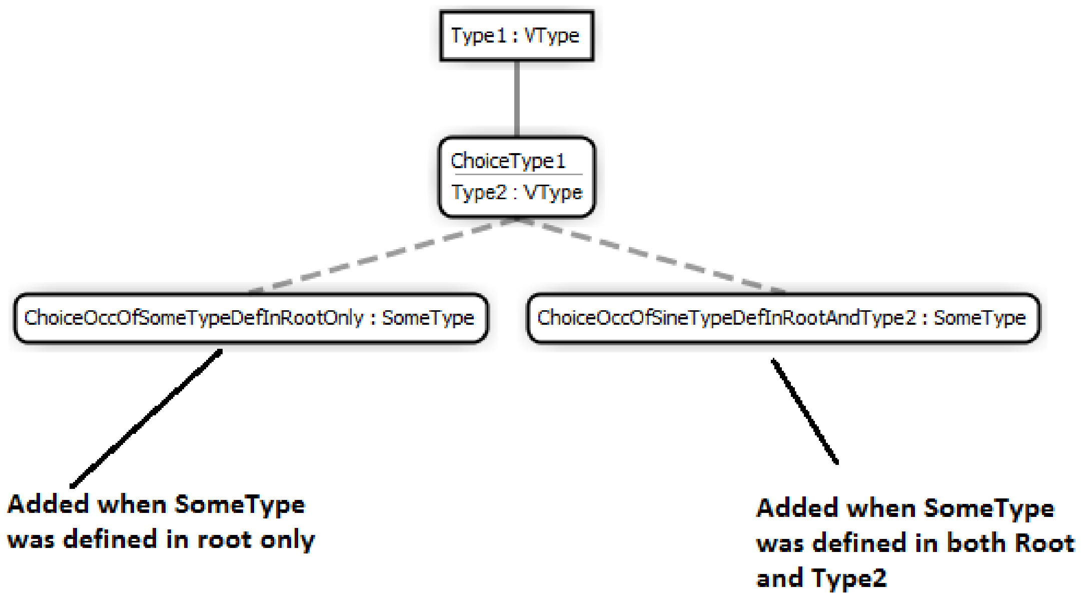


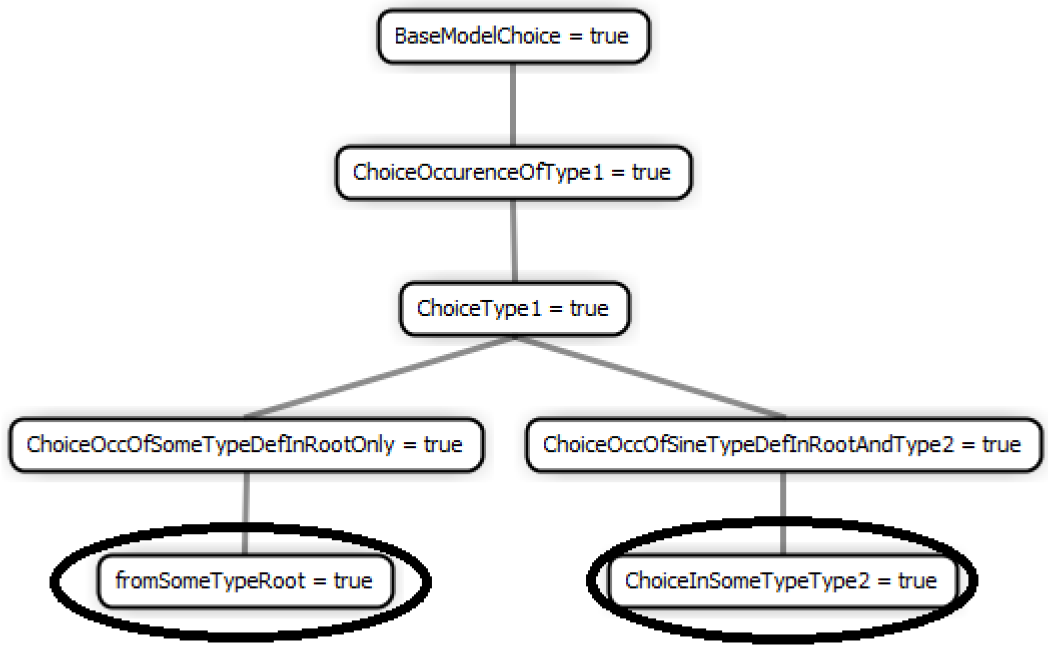
Figure 92 Shadow Problem SomeType defined in Type2

If we finally go back to Type1 and add an occurrence of SomeType as shown in Figure 93 Shadow Problem Type1 with both ChoiceOccurrances, we get the resolution shown in Figure 94 Shadow Problem Resolution.



Type1 : VType
ChoiceType1
Type2 : VType
ChoiceOccOfSomeTypeDefInRootOnly : SomeType
ChoiceOccOfSineTypeDefInRootAndType2 : SomeType
Added when SomeType was defined in root only
Added when SomeType was defined in both Root and Type2

Figure 93 Shadow Problem Type1 with both ChoiceOccurrances



BaseModelChoice = true
ChoiceOccurenceOfType1 = true
ChoiceType1 = true
ChoiceOccOfSomeTypeDefInRootOnly = true
ChoiceOccOfSineTypeDefInRootAndType2 = true
fromSomeTypeRoot = true
ChoiceInSomeTypeType2 = true

Figure 94 Shadow Problem Resolution

The behaviour we have just described is clearly undesirable, and we would therefore suggest that BVR either enforces unique names for VTypes or implements a VType Hierarchy.

6.3 Build procedure

The described procedure will work both with pro-active (variability model is built before production is started), extractive (build variability model from existing production or reengineering products into a production line) or re-active (evolving a single product into a product line).

6.3.1 Suggested building procedure

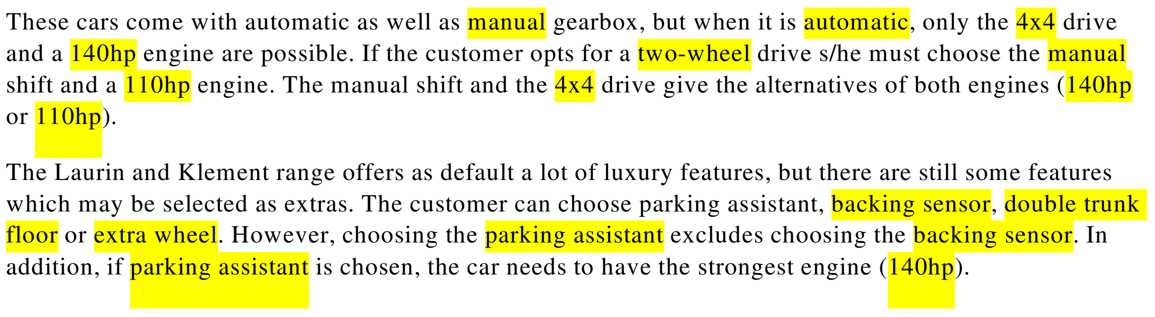
The hierarchical structure of BVR makes it possible to create models, which we can read naturally as choices horizontally and fidelity (detail) vertically.

This model type can be achieved by using the following procedure that we have divided into two parts, identification, and modelling.

Before the task is started, decide on a standard. Decide on the use of name, how to model groups with all optional choices (Mandatory group and "0..*" or optional group and "1..*") etc.

Identification

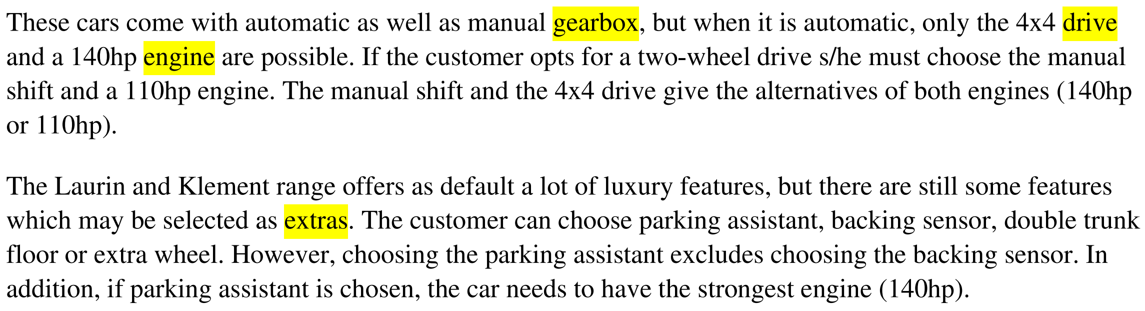
Start by identifying the points of variability, and their type. Feel free to do this in any suitable manner. E.g. if there exists a textual description highlight key words representing variations using different colour for different VSpec types. (In this specification, all VSpecs are Choices)


These cars come with automatic as well as manual gearbox, but when it is automatic, only the 4x4 drive and a 140hp engine are possible. If the customer opts for a two-wheel drive s/he must choose the manual shift and a 110hp engine. The manual shift and the 4x4 drive give the alternatives of both engines (140hp or 110hp).

The Laurin and Klement range offers as default a lot of luxury features, but there are still some features which may be selected as extras. The customer can choose parking assistant, backing sensor, double trunk floor or extra wheel. However, choosing the parking assistant excludes choosing the backing sensor. In addition, if parking assistant is chosen, the car needs to have the strongest engine (140hp).

Figure 95 Identifying variability points

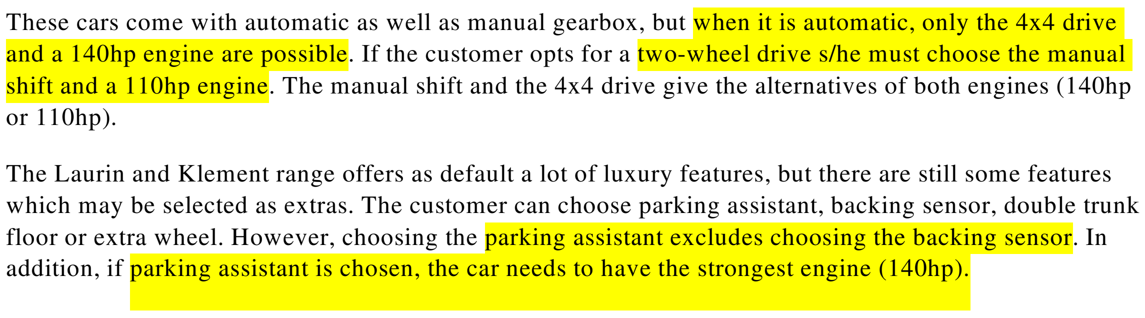
Next step is to identify which collections are required. A collection can be variables that contain higher fidelity choices, parts of the description, or simply groups that the modeller feels are useful. In this step, we also need to identify the Grouping and instance multiplicity for VClassifiers and VClassOccurrences, and if present, we identify VTypes for reuse.


These cars come with automatic as well as manual gearbox, but when it is automatic, only the 4x4 drive and a 140hp engine are possible. If the customer opts for a two-wheel drive s/he must choose the manual shift and a 110hp engine. The manual shift and the 4x4 drive give the alternatives of both engines (140hp or 110hp).

The Laurin and Klement range offers as default a lot of luxury features, but there are still some features which may be selected as extras. The customer can choose parking assistant, backing sensor, double trunk floor or extra wheel. However, choosing the parking assistant excludes choosing the backing sensor. In addition, if parking assistant is chosen, the car needs to have the strongest engine (140hp).

Figure 96 Identifying collections

The last step of the identification process is to find the constraints. For example, two-wheel drive is only compatible with hp110.


These cars come with automatic as well as manual gearbox, but when it is automatic, only the 4x4 drive and a 140hp engine are possible. If the customer opts for a two-wheel drive s/he must choose the manual shift and a 110hp engine. The manual shift and the 4x4 drive give the alternatives of both engines (140hp or 110hp).

The Laurin and Klement range offers as default a lot of luxury features, but there are still some features which may be selected as extras. The customer can choose parking assistant, backing sensor, double trunk floor or extra wheel. However, choosing the parking assistant excludes choosing the backing sensor. In addition, if parking assistant is chosen, the car needs to have the strongest engine (140hp).

Figure 97 Identifying Constraints

Modelling

First, model all the collections related to the current root node. For each collection check and set the correct grouping and optionality. At this point the model of the running example can be seen in Figure 98.

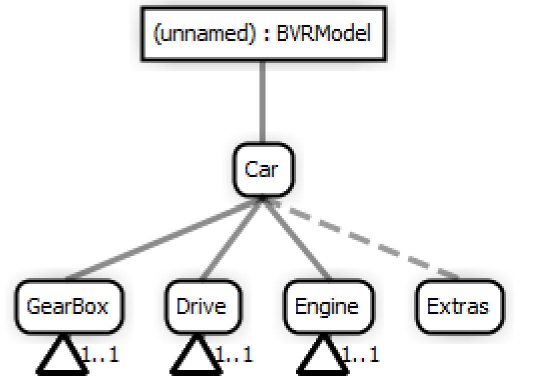


Figure 98 Car model after adding Collections to root

Second, add any variations to this level. Make sure to set the correct optionality as you add them. Since there are no variations to add at this stage, the model is unchanged.

Third, repeat the first three steps to each of the groups, creating the highest fidelity for each group. After this step once the model look like this (see Figure 99)

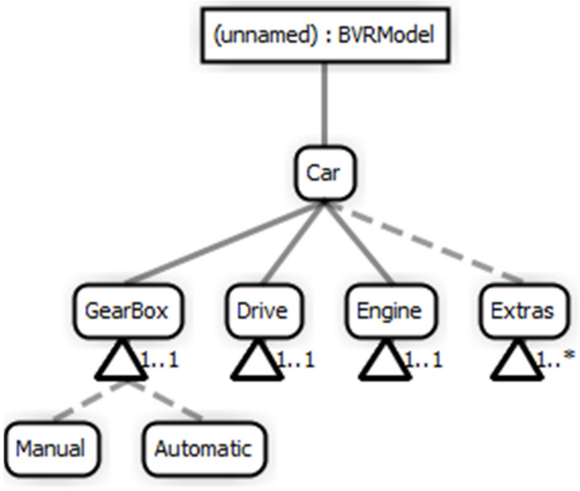


Figure 99 Build procedure after repeating step one and two to the first collection in the model

Running steps one to three for all collections we find the model in

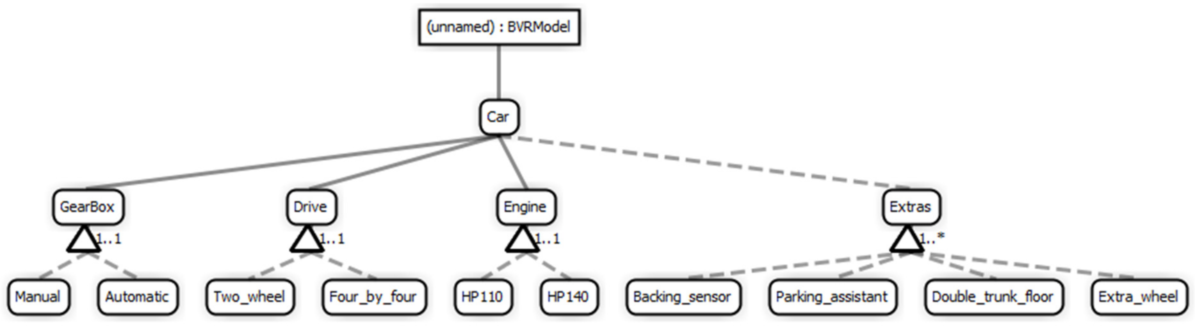


Figure 100 Build procedure after running steps 1-3 on all collections in Car

Finally add the constraints. If a Constraint is only applicable to a single VSpec, add it to the VSpec in question. For all other cases, add it to the VSpec that have all implementations of the VSpec as children. For Modelling task from experiment, the result would look like this:

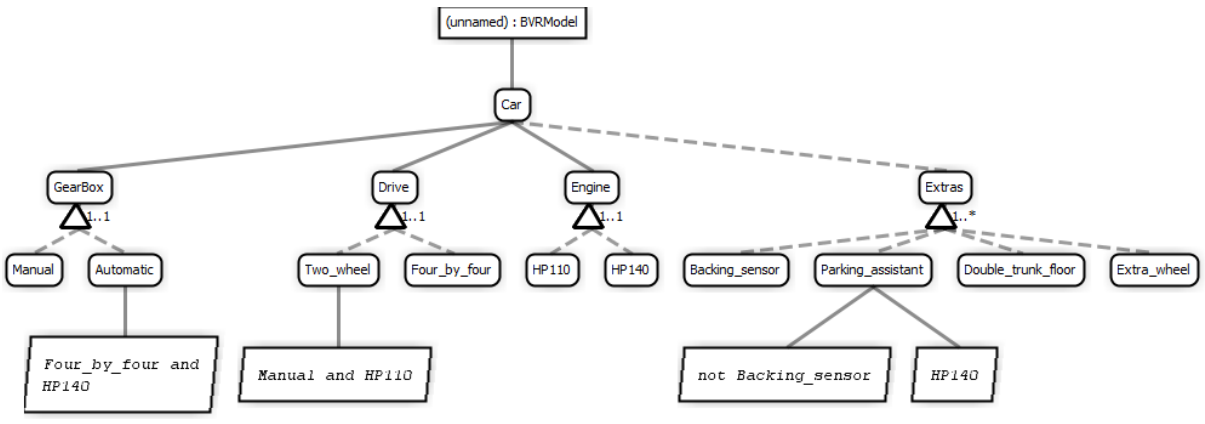


Figure 101 Model after completed build procedure

Process with different VSpecs

In order to both further test the process, and show the results on more complex models we will model specificaitons given in Appendix I Experimental guidelines/Modelling task for procedure test. For this specification, we identify the following:

Variability: account (VClassifier), notifications, sound, vibrate, default, bell, custom, filename (Variable String), sync-time, default, priority, primary, promotions, social, updates

Groups: account, notifications, sound, custom.

Note that VSpect without specified types are Choices.

The model half way through the process would look like this:

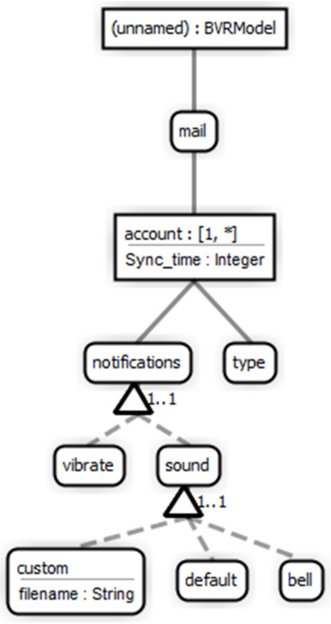


Figure 102 Modelling Task for procedure Test half way through the modelling process

When the process is completed, the models look like this(see

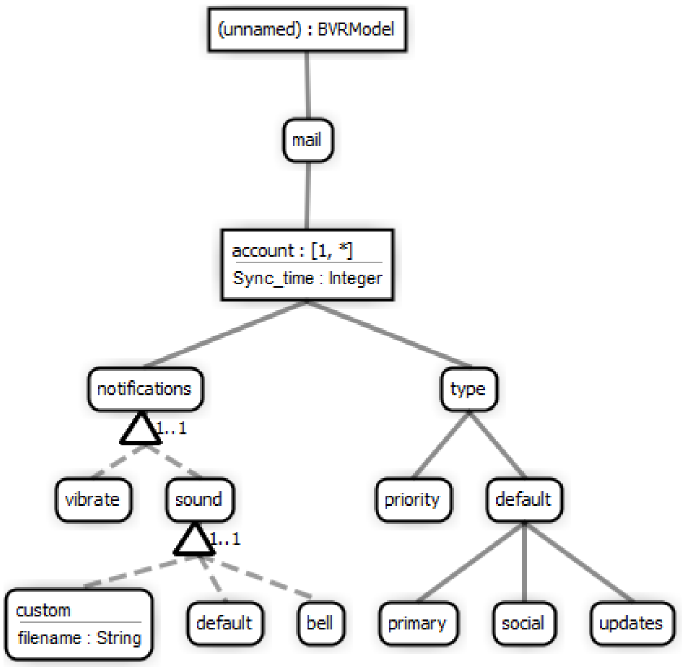


Figure 103Modelling Task for procedure Test at the end of the modelling process

6.3.2 Motivation for suggested procedure

Firstly, this procedure semi-automates the model building, freeing the modeller to focus on making a complete model. Secondly, modelling with increasing fidelity creates models that naturally follows a hierarchical structure of abstraction that helps with complexity management. Thirdly, this modelling process solves the problem of where to build deep and where to build wide, in a manner which should work for most (if not all) cases. Lastly, and most importantly, this procedure focuses on adding all variability points while making sure the modeller sets the correct grouping and optionality. This procedure adds Constraints last, both because cross-model Constraints are impossible to make before all the VSpecs in the Constraint exist (BVR uses strict object creation) and to make it as plain as possible where and when they are needed.

7 Conclusions and future work

In this thesis, we have performed a visual assessment of BVR, which uncovered several problems with the cognitive ease of the BVR language (q.1). Further, we have performed an experiment which uncovered several critical problems regarding the BVR tool (q.3). This experiment also identified several procedures used by the experts while they perform the task (q.2). Lastly, we have suggested several improvements to both the tool, and the language (q.4) as well as a procedure for building BVR models, which we believe help modellers build models, which are RWA and easily understood.

A suggested course of action for future work is to perform three new experiments. The first of these experiments is to repeat the usability experiment from Chapter 5 Expert experience, with a greater number of models and some modifications. First, the experts should all work on the models in the same order. The purpose of keeping the same order is to be able to analyse, and correct for, the learning effect we observed and predicted in 5.3.1 Model/Time spent. The second change is to exclude the resolution stage of the experiment since it in its current form gives too little data, and as such, the resolution process and tool use should be examined separately. The last change is to look for the repetition of behaviour described in this thesis, in order to validate or invalidate the findings.

The second experiment we would suggest is to first implement the changes we have proposed in chapter 6 Proposed improvements. Then have a group of experts and novices perform predefined tasks with and without the described improvements.

The last experiment we suggest is an explicit test of the resolution tool. This experiment would require a large number of different but equivalent VSpec models, for which both novices and experts would create resolutions. We would ask the participants to create specific resolutions, both valid and invalid to see if the found errors. We would ask the participants to create valid models that contain a VSpec resolved in a particular manner (as an example from our running example: “create a car with a parking assistant”). Finally, we would ask the participant to create two or more different and valid resolutions.

Bibliography

(OMG), O. M. G. OMG's MetaObjectFacility.

Atkinson, C., Bayer, J., & Muthig, D. (2000). *Component-based product line development: the KobrA approach*. Paper presented at the Proceedings of the first conference on Software product lines : experience and research directions: experience and research directions, Denver, Colorado, USA.

Bastien, J. M. C. (2010). Usability testing: a review of some methodological and technical aspects of the method. *International Journal of Medical Informatics, 79*(4), e18-e23. doi: http://dx.doi.org/10.1016/j.ijmedinf.2008.12.004

Böckle, G., Pohl, K., & Linden, F. v. d. (2005). *Software product line engineering: foundations, principles, and techniques*. Berlin: Springer.

Boehm, B. (1999). Managing Software Productivity and Reuse. *Computer, 32*(9), 111-113. doi: 10.1109/2.789755

Carter, S., & Mankoff, J. (2005). *When participants do the capturing: the role of media in diary studies.* Paper presented at the Proceedings of the SIGCHI conference on Human factors in computing systems.

DebugMode. Wink. 2015, from http://www.debugmode.com/wink/

Eclipse-Foundation. Eclipse.   Retrieved 310715, 2015, from https://eclipse.org/

Eclipse-Foundation. Eclipse Modeling Framework (EMF).

Gurr, C. A. (1999). Effective Diagrammatic Communication: Syntactic, Semantic and Pragmatic Issues. *Journal of Visual Languages & Computing, 10*(4), 317-342. doi: http://dx.doi.org/10.1006/jvlc.1999.0130

Haugen, Ø. (2012). Common Variability Language (CVL) OMG Revised Submission *OMG document* (pp. 147).

Haugen, Ø. (2014). D4.2 BVR - The language.

Haugen, Ø., & Øgård, O. (2014). BVR – Better Variability Results. In D. Amyot, P. Fonseca i Casas & G. Mussbacher (Eds.), *System Analysis and Modeling: Models and Reusability* (Vol. 8769, pp. 1-15): Springer International Publishing.

Jacob, R., & Karn, K. S. (2003). Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind, 2*(3), 4.

Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A feature-;oriented reuse method with domain-;specific reference architectures. *Annals of Software Engineering, 5*(1), 143-168. doi: 10.1023/A:1018980625587

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study: DTIC Document.

Kirakowski, J. (2000). Questionnaires in Usability Engineering.   Retrieved 18/06, 2015, from http://www.ucc.ie/hfrg/resources/qfaq1.html

Kuter, U., & Yilmaz, C. (2001). Survey Methods: Questionnaires and Interviews.   Retrieved 18.06, 2015, from http://lte-projects.umd.edu/charm/survey.html#2

Lazar, J., Feng, J. H., & Hochheiser, H. (2010). *Research methods in human-computer interaction*: John Wiley & Sons.

Miller, G. A. (1994). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review, 101*(2), 343-352. doi: 10.1037/0033-295X.101.2.343

Moody, D. L. (2010, 2-8 May 2010). *The "physics" of notations: a scientific approach to designing visual notations in software engineering*. Paper presented at the Software Engineering, 2010 ACM/IEEE 32nd International Conference on.

Oracle. Swing.   Retrieved 23.07, 2015, from http://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

Pohl, M., & Scholz, F. (2014). How to investigate interaction with information visualisation: An overview of methodologies (Vol. 8345, pp. 17-29).

pure::variants.   Retrieved 14/07, 2015, from http://www.pure-systems.com/pv/

Specification, O. A. (2006). Object Constraint Language: May.

. Thread safety. (30 March 2015, at 11:22. ed.).

Travis, D. (2009). How to prioritise usability problems. from http://www.userfocus.co.uk/articles/prioritise.html

Trinidad, P., Benavides, D., Durán, A., Ruiz-Cortés, A., & Toro, M. (2008). Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software, 81*(6), 883-896. doi: http://dx.doi.org/10.1016/j.jss.2007.10.030

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages. *Centrum voor Wiskunde en Informatika, 5*.

von der Massen, T., & Lichter, H. (2004). Deficiencies in Feature Models.

Winn, W. (1990). Encoding and retrieval of information in maps and diagrams. *Professional Communication, IEEE Transactions on, 33*(3), 103-107. doi: 10.1109/47.59083

Zhang, G. Quality attributes modeling in feature models and feature model validation in software product lines.

Appendix

Appendix I Experimental guidelines

Experimental procedure

Beforehand:

- Please take as long as you need to acquaint yourself properly with the modelling task (see page 2) before starting the experiment.
- Have the modelling task available while performing the experiment. Either printed out or on a separate screen.
- Create your own Variability model for the given modelling task. These models are used to make sure all experts interpret the modelling task the same way.

Ø: please perform the experiment on the models in ascending order

A: please perform the experiment on the models in descending order

M: please perform the experiment on the models in a 3, 4, 2, 5, 1, 6 order

For each of the models provided:

1. **Start wink and the BVR tool**. Set the wink recording speed to 10 fps.
2. **Move the model into the workspace but do not open it**.
3. **Start the wink recording.**
4. **Open the model in the VSpec editor**
5. **Describe any errors you find in the model in a comment**. (What is wrong? What you believe caused this error? E.g. misunderstood the modelling language, misunderstood the modelling task, misspelling)
   Use the topmost Choice. (or chose one if several choices are at the same level, since there is no way to put comments into VClassifiers)
6. **Use the tool to fix any errors found in as few steps as possible.**
7. **Create two new resolutions**
   - One AWD manual car with parking assistant and double trunk
   - One FWD manual Backing sensor, Double Trunk and extra wheel.
8. **Save and minimize the tool, and stop the Wink recording.**

   In a separate text document called "modelName".txt please:
9. **Write a short description of the error checking and correction strategies you employed**. (if you use the same strategy for several models, write "same as model #")
10. **Write down any appropriate comments to the model**. (E.g. Easier/harder to understand that any others? Did it feel cluttered or clean? Is there anything you would change to make it not only correct but also easier to read?)
11. **Write down any tool help that would have been helpful**.

Modelling task from experiment

Task Description: Skoda Yeti Laurin & Klement

Skoda has a top-of-the-range edition called Laurin and Klement named after the two founders of Skoda, namely Vaclav Laurin and Vaclav Klement.

Our modeling task focuses on this top-of-the-range edition and on its diesel cars.

These cars come with automatic as well as manual gearbox, but when it is automatic, only the 4x4 drive and a 140hp engine are possible. If the customer opts for a two-wheel drive s/he must choose the manual shift and a 110hp engine. The manual shift and the 4x4 drive give the alternatives of both engines (140hp or 110hp).

The Laurin and Klement range offers as default a lot of luxury features, but there are still some features which may be selected as extras. The customer can choose parking assistant, backing sensor, double trunk floor or extra wheel. However, choosing the parking assistant excludes choosing the backing sensor. In addition, if parking assistant is chosen, the car needs to have the strongest engine (140hp).

NB: All VSpec identifiers *must* start with a letter (*hp140* rather than *140hp*)

After you have created the model, please save it as <name>.bvr and export it as <name>.png. Next, please send the two files via email to the instructor (oystein.haugen@sintef.no). The <name> is your real last name.

Once it has been confirmed received, you may leave.

Assignment and models courtesy of Øystein Haugen Senior Researcher, dr. scient. Associate Professor SINTEF

Modelling task for procedure test

In the Gmail, phone app it is possible to add multiple accounts that can be set up with separate settings.

Please create a VSpec of this app with the following settings.

Each account can have notifications, which in turn can be Sound and/or Vibrate, and the sound can be either the Default ringtone, a bell or a custom ringtone in which case a filename must be specified.

The number of days of mail to be synced can also be set separately for each account, along with the inbox type for each. The type can be either default or priority. If the inbox is set as Default, the mail is sorted into Primary and promotions, social or updates.

Appendix II Novice Models

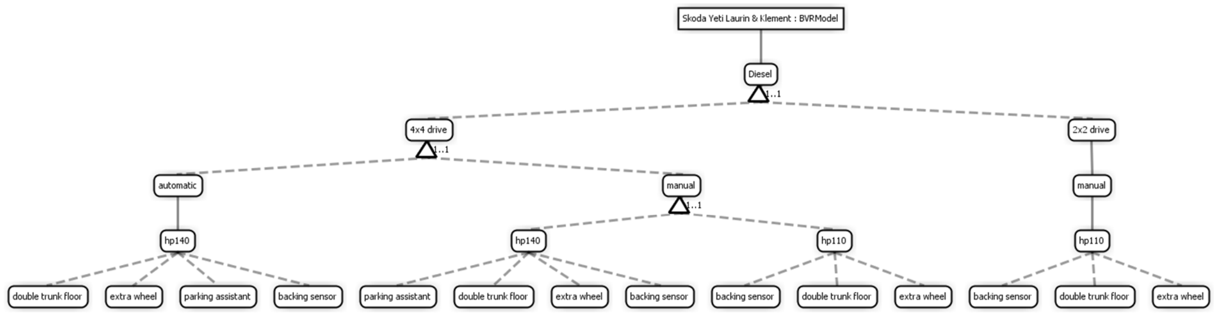


Figure 104 P10

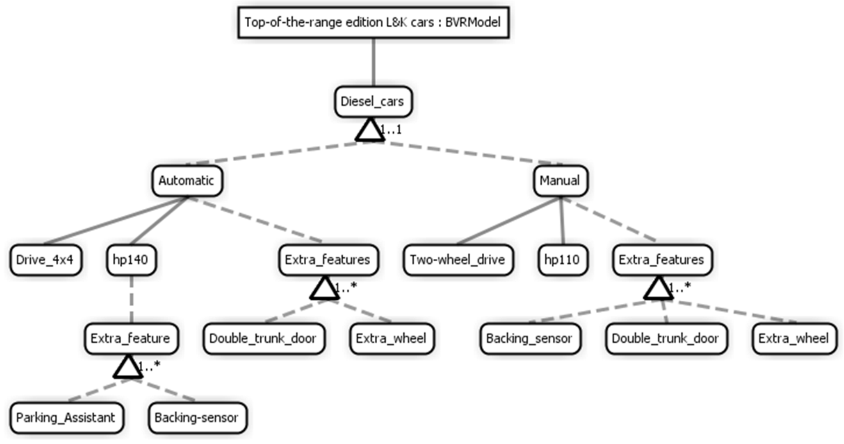


Figure 105 P11

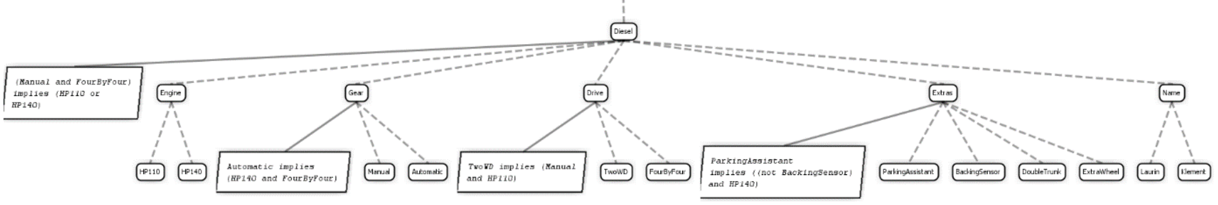

Laurin & Klement : BVRModel
Diesel
(Manual and FourByFour) implies (HP110 or HP140)
Engine
Gear
Drive
Extras
Name
HP110
HP140
Automatic implies (HP140 and FourByFour)
Manual
Automatic
TwoWD implies (Manual and HP110)
TwoWD
FourByFour
ParkingAssistant implies ((not BackingSensor) and HP140)
ParkingAssistant
BackingSensor
DoubleTrunk
ExtraWheel
Laurin
Klement

Figure 106 P12

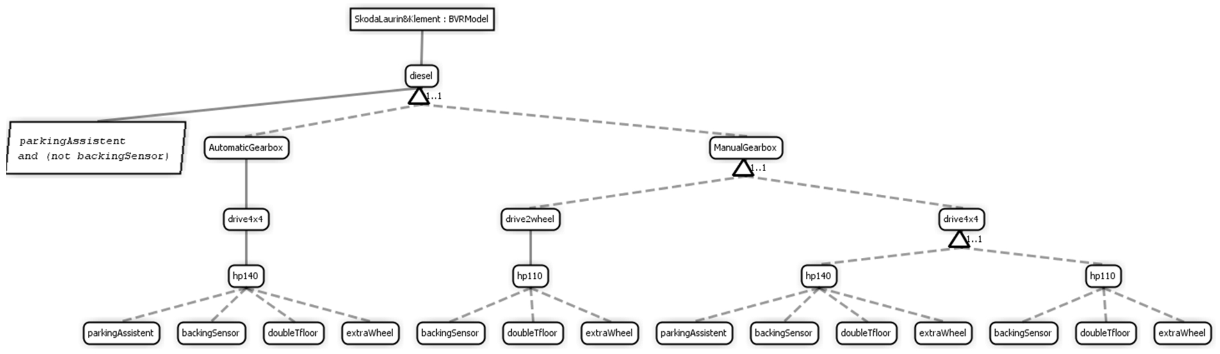


Figure 107 P13

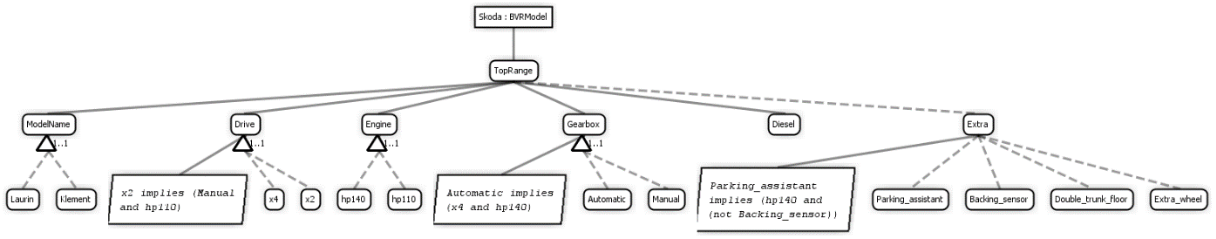


Figure 108 P14

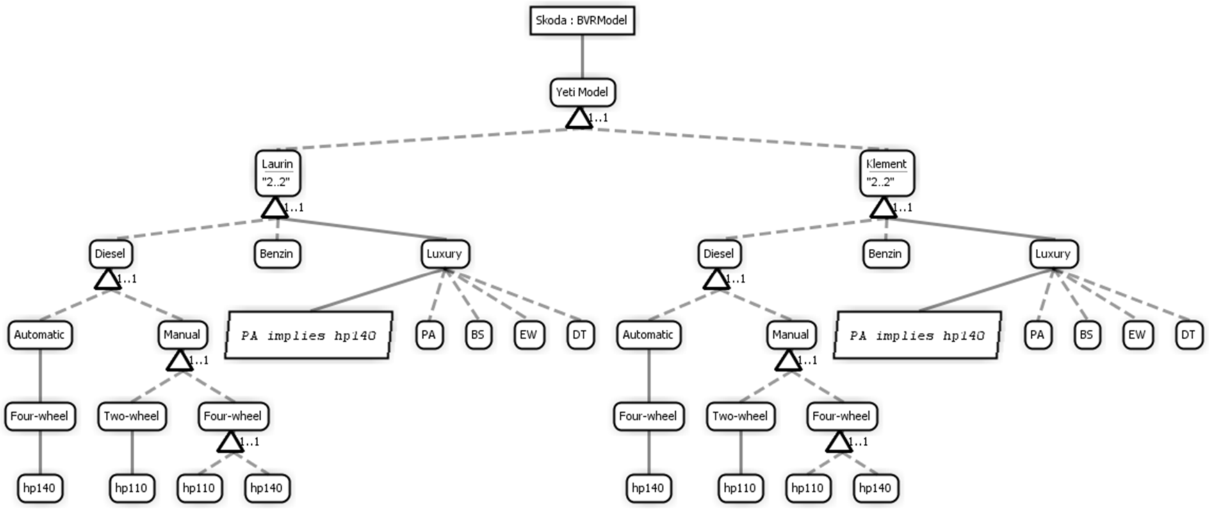


Figure 109 P15