ABSTRACT
During our work on developing and running a software product line for eco-sustainable greenhouse-production software tools, which currently have three products members we have identified a need for extending the notation of the Orthogonal Variability Model (OVM) to support what we refer to as cardinality range dependencies. The cardinality-range-dependency type enables expressing that the binding of a certain number of variants to a variation point can influence variability in other places in the model. In other words, we acknowledge that variability can be influenced, not necessarily by the specific variants being bound, but by their sheer numbers.

This paper contributes with an extension to the meta-model underlying the OVM notation, suggesting a notation for the new type of dependency and shows its applicability. The specific case, which initially required this extension, will work as running example throughout the paper and underline the need for the extension. Finally, the paper evaluates and discusses the general applicability of the proposed notation extension and future perspectives.

Categories and Subject Descriptors

General Terms
Documentation, Design, Standardization, and Languages.

Keywords

1. INTRODUCTION
Software development organizations are met with the challenge of satisfying ever increasing demands for customized software solutions and customized software-intensive systems, which need to be developed with less effort, at lower cost and with a shorter development time, and at the same time with a higher quality. Additionally, they need to manage increasingly complex systems, and create, manage and evolve continuously growing and diversifying products.

The software product line paradigm [11] addresses these requirements by moving software development towards software mass-customization [6], which empowers organizations to develop families of similar software systems at lower cost, in shorter time and with improved quality compared to single-system development. The essence of software product line engineering lays in the ability to exploit the commonality amongst the envisioned products. This is done by handling, localizing and managing the variability amongst the products. However, the success of software product line engineering builds on one more key factor, which is, that the product line members are assembled (or derived or instantiated) from a set of core assets in a prescribed way. This factor distinctions the paradigm from opportunistic reuse, which has been proven inefficient. In this work, we address the documentation, modeling and managing of variability, which quickly becomes a complex, tedious and incomprehensible task, without appropriate facilities in the form of variability models and their specialized expressiveness.

The limitation of the expressiveness of the Orthogonal Variability Model (OVM) was identified through our work on our industrial case study – the software product line called GreenComponents. The software product line is developed to produce a portfolio of software tools, which facilitates growing greenhouse plants in a more energy-efficient and cost-efficient manner. It currently has the ability to produce three individual product members, which are DynaLight Web, DynaLight Desktop and DynaLight Desktop w/control capabilities. The products are currently deployed in the Danish greenhouse sector, and have previously been described in [8, 9, 10]. Furthermore, DynaLight Desktop has previously been exhibited by University of California Berkeley for their industrial partners as an example of sculpting electricity-consumption loads to the immediate availability of electricity.

The SPL is implemented as a NetBeans Module Suite. The module suite contains the source code core assets (structured in modules) to create the three SPL members. The SPL is primarily written in Java and uses the modular architecture and rich client facilities provided by the NetBeans Platform. The NetBeans Module System is used as variability implementation technology and we have previously addressed this subject in an evaluation of the technology as SPL variability implementation technology [10].
New requirements to the variability of our SPL have recently been proposed and we have identified the need for an extended expressiveness of the OVM notation during the analysis of these requirements. The problem concerning the new requirements and OVM is described in greater details in section 3. We will use the modular variability technology described above to implement the requirements in our SPL, but the implementation aspects are not included in this work.

The primary contributions are the identification of undesirable limitations in the expressiveness of OVM and our proposition to extend the OVM notation to remedy these constraints. Despite the fact that we focus almost exclusively on variability modeling in the form of OVM, it should be noted that the most commonly utilized variability modeling language, at the time of writing, is feature models [13], however we choose OVM as modeling language in our project due to the commonly known shortcomings of features models, which we will discuss in section 2. An assessment, comparison and criticism of OVM and feature models is deliberately omitted here as we consider it outside the scope of this paper.

The structure of the paper is as follows: The orthogonal variability model (OVM) is introduced in section 2 to provide common ground in the form of terminology and to describe the current state of the art regarding OVM. The identified problem is described in section 3, where we use our case study to explain the problem and make the deficiency in the expressiveness of OVM more tangible. In section 4, we propose our solution to the problem by providing a new notation and an extension of the OVM meta-model to meet our new requirements. The proposed extension is applied to our problem in section 5. After that we discuss the proposed solution, the consequences and the perspectives, together with future research in section 6, before we summarize the findings and conclude the paper in section 7.

2. ORTHOGONAL VARIABILITY MODEL

The orthogonal variability model (OVM) has been designed to capture variability as a first class concept and to exist on an orthogonal plane with respect to the development artifacts, whose variability it models. It is a centralized model of variability, it is self-contained, and it is separated from the other development artifacts. The model provides a cross-sectional view of the variability cross-cutting the different artifacts involved in software development. The OVM is “a model that defines the variability of a software product line. It relates the variability defined to other software development models such as feature models, use case models, component models, and test models” [11].

The OVM was introduced as variability model in the renowned work of Pohl et Al [11], however, the idea of a separate model to model variability had been proposed earlier [1, 3]. The notation used in OVM is refinements of earlier work, e.g. by the work of Bachmann et al. [1]. The introduction of the new modeling language was intended to tackle some of the shortcomings of feature modeling. These shortcomings have been described by Bühne et Al [3], Bachman et Al [1] and others [4, 11]. Pohl et Al mention some of the disadvantages of modeling variability within the traditional software development models to be [11]:

- The variability is scattered between multiple models, which impedes keeping the information consistent.
- It is hard to determine, which variability information in requirements have influenced which information in design, realization or test artifact.

- The software models, e.g. feature models, are already complex and they get overloaded with additional variability information.
- The definition of variability information within a single development model can often lead to ambiguous definitions of the variability contained in development artifacts.

There has been a significant amount of work on the OVM since it was introduced, e.g. within the areas of notation, automation, techniques, and evaluation [12, 7, 11, 14]. The OVM has a graphical notation defined for its different elements and its structure conforms to its meta-model [11]. The primary entities of OVM are variation points and variants (see Figure 1). Jacobsen et Al first defined these variation points as an identifier of one or more locations in a software asset at which the variation will occur [5].

![Figure 1. OVM notation and meta-model [11]](image-url)
the form of ranges, e.g. [1...5] where it is required that between one and five variants be selected, and [3...7] where three to seven variants need to be selected. The cardinality (referred to as cardinality in [2], and multiplicity and range in [11]) and the proposed dependency constraints [11] are strongly connected to the work described in this paper.

In the OVM, the constraint dependencies between nodes (variation points and variants) are expressed using the graphical notations adhering to the rules expressed in the meta-model (cf. Figure 1). There exist three types of interdependencies, between variation points, between variants (also not related to the same variation point), and between variation points and variants. The dependency types are either ‘requires’ or ‘excludes’ and both are directional in the graphical notation [11]. In other words, feature A may require feature B to function, while feature B does not necessarily depend on feature A to function. The excludes-constraint type has the same semantic according to Pohl et Al [11], however, the excludes-constraint type have also been presented as being bi-directional by Roos-Frantz & Segura [12].

![Figure 2. OVM structured as a hierarchical tree](image)

The organizational structure of OVM is flat, and the specification does not impose a tree-like structure known from e.g. feature diagrams. However, the dependencies between the nodes in the model can create a structure, which can be, but not necessarily is, a tree structure. An example of the structure of an orthogonal variability model is illustrated in Figure 2. The last notational aspect of the OVM is the artifact dependencies, which are the traces to the development assets, covering every artifact that might vary when developing software products e.g. use cases, user manuals, developer instructions, business process, work flows, installation procedures, UML diagrams, software components, software packages, language support, help pages, source files and code segments. There are two distinctive types of artifact dependencies, the variation point dependency and the variant dependency. The first kind of dependency links to the area(s) of the development assets where the variation point is represented, and the second kind links to where the variant is represented.

Optional variation points have been mentioned in the work of Metzger et Al [7] and have been treated in the work of Roos-Frantz & Segura [12]. An optional variation point may be included for a specific product, and does not have to be in the instance of the variability model of a specific product. The mandatory variation point always needs to be represented in a variability model instance, thus there cannot be a valid derivation of the model which does not include the mandatory variation points. The notation of the optional variation point is similar to an ordinary variation point, but with dotted outline. This extension of optional variation point (and the derived mandatory notion) has to the best of our knowledge not been introduced in the OVM meta-model and will not be treated as part of the OVM notation in this paper.

A few additional dependency constraints have been mentioned in the works preceding the introduction of OVM, e.g. the hint dependency which indicates that one variant has some positive influence on another and hinders dependency, which indicates that binding of one variant has some negative influence on another variant [2]. We do not consider these two additional constraint types to be part of the official OVM notation. In general, we consider the work by Pohl et Al [11] to be the first and most important description of OVM and is, therefore, our point of reference.

3. PROBLEM DESCRIPTION

Three different products, each with well-defined scopes and aims to fulfill specific demands in the business plan, are derived from our SPL. The first tool, DynaLight Web, is a publicly available web-solution. It allows growers to upload and analyze their historical production conditions to see how much money and energy they would have saved, if they had used the two other SPL members to guide their production of the past. The aim is to create an incentive to become further involved in the research project and to proceed with utilization of the two other SPL members, DynaLight Desktop and DynaLight Desktop w/control. These two products are desktop applications based on the NetBeans Platform and they both allow the grower to analyze his production locally and to create supplementary light plans. The control feature moves DynaLight Desktop from working as an analysis tool, which helps the grower to plan the use of supplementary light, to become a control system, which is able to autonomously create light plans and execute them continuously until external intervention. In other words, the web-solution provides means to analyze historical conditions to see how much could have been gained by using tools, the next member can analyze the future conditions and create supplementary light plans, and the last SPL member can create light plans and directly execute them and let the process run autonomously.

The software tools sculpt the electricity consumption of the supplementary lighting inside the greenhouses to utilize the electricity more efficiently from a plant physiological perspective and to find the lowest price for reaching a daily production goal. The grower specifies the required production goal in terms of growth, which is measured by the CO₂ absorption of the plants per day, and context-specific properties, e.g. the glass transparency level of the greenhouse, the indoor temperature, and the indoor CO₂ level. The tools find an optimized solution in the form of a supplementary light plan, which is based on the specified growth goal, weather forecasts, hourly electricity prices and predicted hourly photosynthetic gain. The running example concerns the following variability of the tools with respect to the planning algorithm included in the SPL members, whereas detailed treatment of the whole software product line and its members are outside the scope of this paper.

There are currently three different algorithms, which can be applied to solve the planning problem: a greedy optimizing algorithm, a brute-force optimal algorithm, and an optimal pruning algorithm. We will refer to these as Greedy, Brute-Force and Pruning, respectively.

The desktop solutions (i.e. DynaLight Desktop and DynaLight Desktop w/control) are currently deployed with the Greedy algorithm, so the growers have equivalent results from their desktops as they have had using the web-solution. However,
there is an evolutionary pressure to increase the efficiency of the tools, both to provide increased return on investments made by the stakeholders in the research project and to provide an incentive for stakeholders to stay involved. We target these demands by developing an option to include one or more algorithms in the future desktop solutions.

Contrary to the desktop version, we have decided that the web-solution should not have the option of being distributed with the Pruning and Brute-Force algorithms. The reason for not extending the web-solution with these options of the additional algorithms is due to our business plan, where the desktop solutions are available only to stakeholders of the project, and the web-solution is freely available to anyone. There are also some technical reasons behind this decision, but we will not go further into the details, as they do not concern the problem we are trying to outline.

The variability model, which shows that only one algorithm can be selected from the three options including the constraint dependency required by the web-solution, is shown in OVM notation in Figure 3.

The OVM in Figure 3 clearly documents the variability. The variability point VP2 has three variants, V3, V4 and V5, and they can be selected in the range of one to three. In case the web-solution is derived several constraints will apply, as it requires the Greedy algorithm and needs to prevent the two other algorithms from being selected. This OVM is slightly over-constrained and can be simplified by removing one of the two ‘requires’ constraints from the web variant and the two ‘excludes’, but only valid configuration can be derived from the variability expressed in the OVM.

However, further constraints are uncovered when we look closer at the two desktop products, which we will focus on as they exhibit the problem. The planning function cannot use more than one algorithm at a time for creating supplementary light plans, thus an algorithm-selection feature is required in the case an SPL member is distributed with more than one algorithm.

The problem is how to model a dependency on the condition that the number of variants (here algorithms), which are bounded to a variation point, is either above or below or equal to a specific number. And in case the condition is met, then how do we model that another variation point should be included or excluded from the distribution. In other words, it is not the specific variant that triggers the inclusion/exclusion of the algorithm-selection feature, but the number (i.e. cardinality) of selected variants. We have sketched a valid solution in Figure 4.

This solution introduces two almost similar variation points, VP2 and VP3, and several crossover dependencies to enable expression of the new requirements. The model satisfies the OVM meta-model shown in Figure 1 as variants are allowed to be attached to one or more variation points. However, it still introduces two variation points, VP2 and VP3, whose only difference is cardinality, and it further introduces at least three extra dependencies. Refining the model further does not remove these problems (see Figure 5). The problem identified is associated with the limited expressiveness of OVM and does not only apply to our case study, but is widely observable.

In the following are five different scenarios described, each exposing this form of cardinality range dependency.
We propose an extension to the notation of the OVM. The goals for the extension are:

1) To annotate dependencies on the cardinality of variants in a convenient and easily understandable way.

2) To allow a dependency constraint on the cardinality, which is currently not allowed in the OVM notation.

Our new notation uses the following elements:

- A new type of dependency constraint, which is dependent on cardinality, where the selected cardinality is the controlling attribute. The dependency is represented by a directional arrow or by a bi-directional arrow. The directional arrow indicates one-way dependency and the bi-directional arrow indicates mutual dependency.

- A cardinality range descriptor [min...max], which is formed by using \([0...n], [m...n], [0...*], [m...*]\) where \(m, n \in \mathbb{N}\) and * means no upper bounds, the left number is minimum cardinality and the right number is maximum allowed cardinality of the range.

- Four textual descriptors that unambiguously capture whether a dependency relates to a variation point or a variant, and if the cardinality should be constrained by excluding or requiring. The four types are: requires_v<range>, excludes_v<range>, requires_v_range, and excludes_v_range, where <range> should be substituted with the range descriptor from the second bullet point.

At the same time we also propose an extension to the meta-model that is part of the extension of the OVM notation proposed above. The extension is treated separately from the rest of the OVM standard meta-model and is shown in Figure 6.

The alternative choice, variation point and variant elements are part of the standard meta-model, whereas the abstract notions of variation point cardinality constraint dependency and variant cardinality constraint dependency, and the four elements: requires_vp_range, excludes_vp_range, requires_v_range and excludes_v_range are additions to the original model.

There can be one or more dependencies between an alternative choice and variation points, and variation points can be dependent on several different alternative choices. In case there is a dependency between a variation point and an alternative choice, then it must be either the type requires_vp_range or excludes_vp_range. No matter which one is instantiated it must have a minimum and maximum specified that describes the range.

In the same way there can be one or more dependencies between an alternative choice and variants, and variants can be dependent on several different alternative choices. In case there is a dependency between a variant and an alternative choice, then it must be either the type requires_v_range or
excludes_v_range. Also in this case there must be specified a minimum and maximum for the range.

<table>
<thead>
<tr>
<th>Constraint Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires_V_[min..max]</td>
</tr>
<tr>
<td>excludes_V_[min..max]</td>
</tr>
<tr>
<td>requires_VP_[min..max]</td>
</tr>
<tr>
<td>excludes_VP_[min..max]</td>
</tr>
</tbody>
</table>

Figure 7. Extension to notation

The extension of the original meta-model could have been modeled in several alternative ways, and while some of them might result in a simplified meta-model, we have chosen to model the extension in the same style as the original meta-model. An example of a more simplistic meta-model could combine the two dependency types into one cardinality dependency, which was not segregated into two by its relation to variant or variant points.

However, we will argue that the two distinct types of dependencies help prevent misinterpretations and misunderstanding when reading the model, because of the textual tags that describe the belonging to either a variant or a variation point.

Moving from the meta-model extension to the notational extension we propose the four new dependency constraints shown in Figure 7.

The new extension simplifies and enhances the clarity of OVM, removing the need for superfluous variation point and dependencies, which are introduced by the problem described in section 3. We will now demonstrate the ability and underline the before-mentioned quality enhancements on our case study.

5. APPLICATION OF THE SOLUTION

We have shown our proposed solution on the running example in Figure 9, which is semantically equivalent to the model we developed in Figure 5. The syntax, however, is changed.

Modeling the requirements using the extended notation removed the need for one alternative element, one variation point, and three optional dependencies, one requires_vp_vp and one excludes_vp_vp dependency constraint, while it gained one one-directional requires_vp_range constraint.

In order to further illustrate the applicability of the extended OVM and compare it to the expressiveness of the standard OVM notation we have chosen to include one additional example based on the following requirement.

**Req.1**: “…when there is no more than one algorithm, there is no need for a selection mechanism, but when there is more than one algorithm there should be a manual selection mechanism, while when there are more than three there should be a semi-automatic guiding selection mechanism”.

The variability encompassed in the requirement is modeled using the standard OVM notation (see Figure 8(A)) and using the proposed extension to the OVM notation (see Figure 8(B)).

The model in Figure 8(A) contains four variability points and five variants, while Figure 8(B) has two variability points and five variants. The number of variation points has been reduced by 50 percent, from four to two, whereas the number of variants is unaffected.

Looking at the optional dependency between variation points and their respective variants, we see that Figure 8(A) has eleven optional dependencies, while Figure 8(B) has five. In other words, there is a reduction of 65 percent, from eleven to five, in the amount of optional dependencies used to describe the same requirement. The alternative elements have also dropped 50 percent from four to two.

![Figure 8. Comparison of standard OVM notation (A) and the proposed extended OVM notation (B)](image-url)
FIGURE 9. Proposed OVM extension applied

The last comparison of the models in Figure 8 concerns the number of dependency constraints involved to express the requirement (cf. Req.1). Here the model in Figure 8(A) has five dependency constraints, between VP1 and VP2, between VP1 and VP3, between VP1 and VP4, between VP3 and V1, and between VP4 and V2. While the model in Figure 8(B) has four requires dependency constraints, between VP2’s cardinality and VP1, between VP2’s cardinality and V1, between VP2’s cardinality and V2, and a single excludes cardinality range constraint between VP2’s cardinality and VP1. The differences between the two models in relation to their elements and connectors are shown in Table 1. The two comparable models from the case study are also included in the table.

TABLE 1. Number of Elements in the two couples of models

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 5</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Figure 9</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Figure 8(A)</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>Figure 8(B)</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>18</td>
</tr>
</tbody>
</table>

6. DISCUSSION

In this section, we discuss the proposed extension to the OVM notation using a set of questions.

• Does the extension of the OVM notation make the variability models simpler and, thus, more comprehensible?

We observe that the number of elements used to represent the same requirements is significantly reduced when looking at the Table 1. The variability models which use the proposed extension are obviously simpler, in our case, compared to the original models, because there are fewer elements and connectors for the reader to grasp (cf. Table 1). This makes the models using the proposed extension more comprehensible, and does increase the understandability of the requirements being modeled. The following requirements “when there is more than two variants there should be a selection mechanism included” is, in our opinion, definitely more intuitively read from the new model (cf. Figure 9) using the proposed notation, than when trying to read the model shown in Figure 5.

We further back these claims by showing an example of the requirement of Req.1. In this hypothetical, but realistic scenario the simplification facilitated by the extended expressiveness of the modeling language is even clearer, and underlines the advancement of the contribution in our work.

However, we could have substantiated our statement regarding the enhanced comprehensibility by e.g. performing questionnaires on a group of developers, but it should be acknowledgeable by studying the models (cf. Figure 5, Figure 9, and Figure 8) that this improvement in comprehensibility is plausible. We argue that, if we assume that fewer elements in a model make a model simpler, and that simpler models are more comprehensible, and that it is the proposed extension that enabled the reduction of elements in the model, then we can deduct that this expressiveness enabled by the extended notation can make models simpler and more comprehensible.

The proposed extension comes at a cost of increasing the complexity of the modeling language. The addition of four new constraint dependencies increased the number of constraint dependencies from six to ten, which is a substantial increase. It is however our opinion that these additional constraint dependencies do not increase the adoption barrier for newcomers of the OVM language significantly and the advantages of the gained expressiveness outnumber the downside.

• Were the goals for the proposed solution satisfied?

The proposed extension to the OVM language allows us to annotate dependencies on the cardinality of variants. The convenience and understandability, is, however, more difficult to evaluate. The notation has been very easy for us to adopt, and we find it to be an intuitive extension to the language, which is both convenient and easily understood.

• What are the limitations of the proposed extension?

Elaborating on the problem, we identified other scenarios that extended the problem observed in our case. We move towards the restrictions of choosing a single range to depend on. Imagine a case where a variation point is dependent on multiple sets ((1..3), (5..7),(9..11)). This form of dependency cannot be expressed as one dependency using our extended notation. However, this can be captured by creating multiple cardinality range dependencies, which is allowed according to our extension of the meta-model.

Another limitation is that OVM, even with the extension, cannot express conditional expressions like e.g. (x modulus 2 == 0), which e.g. could enforce that a third-part negotiation mechanism should be included if an even number of variants were included.

However, we have not identified the need for supporting conditional expressions so far and we have difficulties coming up with realistic examples. Because of the lacking realistic requirements for these types of extension the conditional expression dependencies have been transferred as ideas to our future research, as we believe that a language should not be extended more than there is a need for.

• Does the proposed extension negatively influence the work already done on automation and formal language description?

The new notation can be transformed to adhere to the standard OVM notation. Every cardinality range dependency constraint can be expanded to a variation point with an alternative range specified by the multiplicity, and have constraint dependencies added. This means that the work already done with respect to the standard OVM notation does not likely need to be modified as a consequence of the proposed extension.

7. CONCLUSION

The work described in this paper contributes to the field of variability modeling in several different ways, but the primary contribution is the proposed extension of OVM which enables it to express dependencies on the cardinality of alternative choices. It is our hope that this contribution will be adopted by the practitioners and researches to create simpler and more comprehensive models, which again will help managing the ever-increasing complexity of systems with variability.

Additionally, this work contributes with summarization of the work concerning OVM from its introduction to the current state, the description of the case study wherein the issue with
the lacking expressiveness was identified, together with the extension of the notation and its meta-model, the illustration of the applicability and exemplification of the proposed extension in conjunction with the original notation, and finally the discussion of the results, limitations and perspectives.

8. ACKNOWLEDGMENTS
Our thanks go to the University of Southern Denmark and the project ‘Intelligent Energy Handling in Greenhouses’ which is funded by Region of South Denmark and the European Regional Fund.

9. REFERENCES