Understanding Decision Models – Visualization and Complexity reduction of Software Variability

Thomas Forster, Dirk Muthig, Daniel Pech
Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhofer-Platz 1, D-67663 Kaiserslautern, Germany
{forster, muthig, pech}@iese.fraunhofer.de

Abstract
With the increasing size and complexity of software systems also the amount of software variability grows. In this paper we present decision models as a means of dealing with software variability and views on decision models that are supposed to make the large amount of variability manageable. Also some mechanisms for supporting the process of decision modelling and resolving decision models are introduced. In a final experiment we evaluate the presented views and process support mechanisms.

Keywords: product lines, variability management, views, decision model, Decision Modeller, empirical study

1. Introduction
Variability has ever been an inherent property of software, which arises from the need of developing software artefacts for the deployment in different contexts. This requirement necessitates the development of highly adaptable software artefacts. The locations at which a software artefact can be extended or configured for a particular context are so-called variation points. They make up a software component’s variability. This variability has to be managed somehow, nowadays more than ever as a result of the rising amount of variability. There are two major reasons for the increasing software variability. The first one is the development of software product lines. There design decisions are left open intentionally and are postponed to a binding time as late as possible in the software development process. The second reason is the relocation of functionality from mechanics and hardware to software. This phenomenon can particularly be observed in domains using embedded systems, as for instance the automotive or avionic domain. [1]

The huge amount of variability brings up new challenges according the management of those. In order to design the variability management more efficiently several requirements addressing qualities, such as usability, testability, scalability, traceability, must be considered. In this paper we will focus on scalability and traceability, which interleave with understandability and reduction of complexity of decision models:

- Scalability in our context means that it must be possible to handle rather large and complex product lines.
- Traceability means in our context that it must be possible to keep variabilities consistent and maintainable. Thus, it must be possible to establish dependencies between variation points that realize certain variability in software artefacts of different development phases.

The remainder of this paper is organized as the following. In the next section we introduce the concept of decision models. Section 3 then discusses different views on decision models. In section 4 a tool, that implements the suggested concepts, is presented. The tool is then validated in a small case study presented in section 5. Finally section 6 concludes this paper.

2. Decision Models
Because variability plays an important role in current-day software, we need a model to explicitly document and manage this variability.

The two main approaches that are applied for modelling variability are feature models and decision models. However, this paper’s focus is on decision models, thus we give a brief introduction to those.

A decision model is defined as a model that "captures variability in a product line in terms of open decisions and possible resolutions. In a decision model instance, all decisions are resolved. As variabilities in
generic work products refer to these decisions, a
decision model instance, also called resolution model,
defines a specific instance of each generic work
product and thus specifies a particular product line
member”. [3]

Basically a decision model is a table, whereby each
row in the table represents a decision and each column
a property of a decision. A decision has the following
properties:

- **ID**: A unique identifier for the decision (usually an
  integer value)
- **Question**: A question which makes the decision
  more understandable when deriving a decision
  resolution model.
- **Variation Point**: The point in an asset which is
  affected by this decision.
- **Resolution set**: A set of answers to the decision’s
  question.
- **Effect**: An effect for each possible answer to the
  decision’s question (constraints).
- **As necessary, further properties can be added.**

The KobrA method sub-divides decisions into two
types, simple decisions and complex decisions. A
simple decision directly affects a product line asset and
does not affect any other decisions. [4]

An example for a complex decision is decision 2 in
Figure 1, which shows an exemplary decision model
for a coffee machine. If the decision is resolved with
coffee slot, decision 4 is resolved with no. This is due
to the reason, that it makes no sense to install a
container for coffee-beans, if there is no mill which
can crush the beans.

Answering all questions and thus resolving all
decisions leads to a resolution model. A resolution
model (configuration) consists of all decisions and the
answers to their questions. That is, a resolution model
constitutes a concrete member of a product line.

### 3. Visualization mechanisms

Since human beings can assimilate complex
coherences easier when those are visualized somehow
we come up with a metamodel and graphical notion for
modelling constraints among decisions. For modelling
constraints we use logical expressions as already
proposed by Schmid and John who use those to
formulate constraints in their approach [5] or xADL
which uses boolean expressions to model constraints in
the form of boolean condition guards. [6]

Furthermore, we introduce several views on
decision models which are used to reduce the
complexity and to focus on particular aspects of a large
model. In order to further structure the huge amount of
decisions within a view, resulting from large and
complex projects, we bring in the concept of layers.

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Variation Point</th>
<th>Resolution</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Can cups be warmed up?</td>
<td>Cup warmer</td>
<td>yes</td>
<td>A warming plate is attached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>No warming plate is attached</td>
</tr>
<tr>
<td>2</td>
<td>Does the machine have a crushing mill for coffee, a slot for putting in coffee powder or both?</td>
<td>Input</td>
<td>crushing mill</td>
<td>A crushing mill is attached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>coffee slot</td>
<td>A slot for coffee powder is installed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>both</td>
<td>A crushing mill and a slot for coffee powder is attached</td>
</tr>
<tr>
<td>3</td>
<td>Does the machine have a milk frother or a cappuccinatore?</td>
<td>Milk frother</td>
<td>milk frother</td>
<td>A milk frother is attached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cappuccinatore</td>
<td>A cappuccinatore is attached</td>
</tr>
<tr>
<td>4</td>
<td>Does the machine have a container for coffee beans?</td>
<td>Bean container</td>
<td>yes</td>
<td>A container for coffee beans is attached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Resolve decision 2 with: crushing mill or both</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>No container for coffee beans is attached</td>
</tr>
</tbody>
</table>

*Figure 1: Exemplary decision model*
3.1. Constraint Modelling View

Metamodel
The metamodel for logical constraint expressions is depicted in Figure 2 and Figure 3. A logical constraint expression is a hierarchy of operators and operands. In more detail a logical constraint expression can consist of an implication or any other operator as top level node. An implication, as in predicate logic, is an operator always consisting of two operands, a premise and a conclusion. A conclusion constitutes facts, which become effective if the premise (assumption) is fulfilled. Premises and conclusions in turn consist of other operators.

Figure 2: Metamodel for implications

![Figure 2: Metamodel for implications](image)

Figure 3: Metamodel for operators

Following syntactic rules must be adhered to when instantiating the metamodel:
- An implication always contains exactly one premise and one conclusion
- Binary operators must contain exactly two operands.
- Unary terms must contain exactly one operand.
- Assignments can only be contained as leafs.
- Elements already referenced in an implication’s premise cannot be referenced in the implication’s conclusion or the other way around.

Notation
As graphical notation for constraint expressions we choose a notation similar to that used in digital technology for logic gates. The graphical representations are shown in Figure 4. However, as digital technology does not contain something similar to an implication, a new icon for this has to be introduced. An implication will be represented by a circle containing an arrow. The representation of an assignment in a logical expression (which reflects a decision in the decision model) is a rounded rectangle as shown in Figure 5. The rectangle consists of two compartments, the head compartment which contains the decision’s name and a second compartment containing the domain values the decision is to assume.

Figure 4: Logical gates

![Figure 4: Logical gates](image)

Figure 5: Decision

![Figure 5: Decision](image)

To create a constraint, the decisions and gates are then connected using arrows as in the constraint illustrated in Figure 6.

Figure 6: Exemplarily constraint

![Figure 6: Exemplarily constraint](image)

The meaning of the example constraint is that, “if decision dec1 is resolved with the value show and decision dec2 is resolved with the value hide, then decision dec3 must be resolved with the value show”. On the one hand the direction of arrows prescribes the direction of how to read the constraint and on the other hand it defines the relationship between the connected items, i.e. which element is the operand and which the operator. An arrow connecting two items referenced in a premise designates the source element as and operand and the target element as the operator. Arrows in a conclusion must be interpreted the other way around. This way of modelling constraints always results in a tree like structure. Thus, if an implication is contained in the expression, that implication builds the centre of the model to which sides respectively expands a tree of gates and decisions.

3.2. Dependency View

A further view that helps to keep track of large and complex decision models is the dependency view
[7][8]. It abstracts which decisions in a decision model are related to each other, but not how exactly. Consequently it can be seen of as an abstraction of all existing constraints.

**Metamodel**

The dependency view is based on the metamodel shown in Figure 7. The creation of constraints among decisions in a decision model results in a set of relations because a constraint usually references several decisions. Therefore, the relationship element captures only the knowledge of how two decisions (a source decision and a target decision) are related, i.e. which one has an effect on the other one, but not what exactly that effect is.

Figure 7: Metamodel for dependency view

**Notation**

One possibility of representing the dependency view is as a node- and edge-based view. Decisions are represented by a rounded rectangle that contains the name of the according decision.

Relations between decisions are shown as simple arrows that connect the related decisions and therefore visualize which decisions influence each other. The direction of the arrow indicates if a decision is influenced by another decision or if it influences another decision. The arrow’s source decision is the affecting decision, whereas the target decision is the affected one. However, from what exactly the influence results and what preconditions must be met in order to fire can not be seen in this view. An example for a dependency view can be seen in Figure 8. It shows for example, that decision 1 affects two decisions, namely decision 2 and decision 3. This influence is the result of a constraint with the name c1, which is indicated by the label next to the arrow representing the according constraint. Another thing to see is that decision 6 is influenced by decision 3 and decision 4. The two influences have different reasons, firstly constraint c3 and secondly constraint c4.

A second way of representing the dependency view is text based. Therefore, a table consisting of five columns is used as shown in Table 1. The first column shows a decision’s name, the second column shows by which other decisions it is influenced. Column three adds the number of decisions affecting the given decision. The fourth column then indicates, which other decisions are influenced by the given decision. Again this number is added in the fifth and last column of the table.

![Figure 8: Graphical representation of the dependency view](image)

![Table 1: Tabular representation of the dependency view](image)

A last option of representing the dependency view is emphasizing the hierarchy of decisions influencing each other. This can be compared to a call hierarchy in a programming language like java for instance. Accordingly, for a decision model the view focuses on a single decision and either shows which decisions affect this decision or which other decisions are affected by this particular one. Figure 9 shows an example for such a hierarchy, in which the starting point is decision 1. It can be seen, that decision 1 affects decision 3 and that in turn influences decision 6. Decision 6 however does not affect any further decision, as it is the hierarchy’s endpoint.
3.3. Layered view

Each of the previously presented views can be enriched by using layers to further structure the contained elements.

An example for the application of layers could be the mapping to development phases (e.g. analysis, design, implementation, testing). Decisions in a layer then only affect decisions in a lower layer or product line assets of the according development phase.

Metamodel

The metamodel for layers is depicted in Figure 10. A layered view on a decision model consists of an arbitrary number of layers (configurable by the user). Those layers in turn consist of compartments that introduce a further way to structure decisions within a layer. Finally the compartments contain links to concrete decisions of the decision model.

3.4. Filtering

A useful concept to reduce a decision model’s complexity is filtering. That is, the masking of certain representation elements based on some property. Such a property for filtering could be the information of a decision to which stakeholder that decision is relevant. For example, decisions representing features (i.e. top most decisions) are usually of interest to people responsible for requirements and the customer, whereas, decisions representing variability on design or code level (i.e. lowest decisions) are only of interest to developers. A property for filtering could also be the information if decisions are simple decision or a complex decision. Applying such a filter to a view would then look like in Figure 12. The view on the left of the figure shows an unfiltered dependency view, after applying the filter for simple decisions only decision 5 and decision 6 are still visible, because they are the only simple decisions in the view (since they don’t affect any other decisions).

3.5. Resolution Processes

An issue in the process of product configuration, for decision models with a large number of decisions, is the selection of an appropriate starting point for the configuration and the order in which decisions are traversed and resolved. That especially applies to people who did not develop the decision model and thus have no knowledge about its structure. Therefore,
the domain engineer could create a process, which guides application engineers in resolving the decision model. That is, the process consists of a well-defined order of how to resolve decisions. The process creation could be accomplished automatically by using particular strategies. Such a strategy could be that decisions at the top of a dependency hierarchy have to be resolved first, then the decisions deeper in the hierarchy. Another possible strategy, based on a layered view, could be to first resolve decisions in the top layer, then the ones in the next layer and so on. There might exist other strategies how to find a resolution process, but this is not elaborated on here, because it is out of the scope of this paper.

4. Tool support

As described in section 2 variability and variation points can be managed using decision models. In principle it is possible to handle decision models manually, for instance by using Excel sheets, which map variation points on artefact elements and decisions for instantiation of the model (see in Figure 13 for an example). The resolution of a decision can constrain other decisions (Decision 8 is resolved to “Yes” if Decision 1 is resolved to “No”). Constraints, consistency problems with large artefact models and their associated decision models as well as the instantiation process lead to the development of tool support in form of the Decision Modeller. The tool is described in more detail in [12] and basically served as a proof of concept for [13].

<table>
<thead>
<tr>
<th>Question</th>
<th>Resolution</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Should the wiper stop after ignition?</td>
<td>Yes</td>
<td>- Remove stereotype variant from FinishWiping</td>
</tr>
<tr>
<td>No</td>
<td>- Resolve decision 8 to yes</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8 A second wiper present?</td>
<td>Yes</td>
<td>- Remove class ...</td>
</tr>
<tr>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 13: Decision Table

The conceptual architecture of the Decision Modeller tool is shown in Figure 14. As the Decision Modeller is realized as a set of Eclipse plug-ins, the architecture primarily follows a Model-View Controller style. There are five conceptual components:

- **UI:** The UI component provides the graphical user interface elements necessary to work with the Decision Modeller tool. This includes a set of tree viewers for displaying the model as well as wizards to create Decisions, Constraints and projects.
- **ExternalTool:** The ExternalTool component represents external modelling tools (e.g., Rational Software Modeler, Visio) that can be extended by the Decision Modeller to manage variability and to resolve constraints and instantiate variants in these external tools. The Decision Modeller provides standard interfaces for its features and resolution functionality.

Figure 14: Decision Modeller – Conceptual Architecture

- **Modelmanagement:** The Modelmanagement controls the life cycle of the in-memory data models at runtime. Moreover, it is a façade to the components Algorithms and Model, which should not be accessed directly and not communicate with each other.
- **Model:** The Model component encapsulates the internal data model of the Decision Modeller.
- **Algorithms:** The Algorithms component realizes a reasoning engine for constraint evaluation.

For validation purposes we implemented the concepts listed in section 3 as a contribution in terms of a visualization component to the Decision Modeller.

5. Validation

In order to validate the concepts from section 3 we conducted an experiment based on the Goal-Question-Metric method (GQM) [11].

5.1. Goal

The goal of the experiment was to understand if the implemented concepts help to reduce complexity and improve scalability and traceability of decision models.

5.2. Question

To operationalise the goal following questions are defined:
• **Q1**: How long does it take the attendees to accomplish a certain task referred to modeling or understanding?
• **Q2**: How many faults do the attendees produce in the task?

5.3. **Hypotheses**

According the questions following null hypothesis can be formulated:

\(H_0\) – **There is no difference between tool support with and without visualization component, neither with respect to usability nor with respect to the reduction of complexity or improvement of scalability and traceability of decision models.**

Due to the expected observations three further hypotheses can be formulated:

- **\(H_1\)** – The concepts implemented by the visualization component reduces the complexity of decision models.
- **\(H_2\)** – The concepts implemented by the visualization component improve the scalability of decision models.
- **\(H_3\)** – The concepts implemented by the visualization component improve the traceability of decision models.

5.4. **Metrics**

To collect the data needed to answer questions Q1 and Q2 the following metrics were used:

- **Time** needed to accomplish a task.
- **Number of faults** made in the task. Due to time constraints it was not possible to work out an explicit definition for faults. Therefore, the decision what exactly stated a fault was made by the person who evaluated the results of the experiment.

With this data collected, conclusions about the effort spend on a solution and the efficiency of a solution can be drawn. The correlation between the two metrics and a solution’s efficiency is depicted in Figure 15. In case a task is solved in a short time and with no or almost no faults, the task’s solution was developed efficiently. If the task is accomplished with a few faults and a short time or the other way around, with almost no faults and in no short time, the solution has a medium degree of efficiency. If the number of faults or the needed time exceeds a particular boundary, the solution is inefficient.

5.5. **Setup**

The participants in the experiment were split into two groups, each consisting of 3 students of computer science. The first group had to accomplish a set of given tasks with a version of the Decision Modeller that did not have the additional visualization component. The second group had to process the same set of tasks, but with a version of the Decision Modeller that contained the visualization component. Both groups were given the same experiment description document and a short introduction to feature and decision modelling, so they had a common base of knowledge. Of course, the tool introduction was slightly different for both groups, as the tool setup was different. The first part of the document contains an initial questionnaire to determine the subjects’ experiences in modelling and modelling tools. This was important, because a higher experience in this field would reduce the time needed to accomplish some tasks. The document’s second part contained three tasks and their description.

- The first task was about modelling constraints for the decision model of a coffee machine. As input the subjects got a decision model, already containing all simple decisions, and the coffee machine’s feature model, as well as its detailed description. This task addresses scalability, because the decisions had to be related to each other. For this purpose the correct decisions and according values had to be found in the model.
- The second task was to extract information from a given decision model. For this task the subjects did not have to model anything. This task also addresses scalability, because the necessary information had to be found in the model.
- The third and last task required the subjects to write down a possible resolution process for a
given decision model. Here traceability is addressed, because it was required to retrieve an understanding of how the decisions are related to each other.

Before and after processing a task the subjects had to write down the actual time in order to determine the task’s duration. For counting the number of faults, also the subjects’ workspaces, containing their outputs produced in the experiment, were collected.

5.6. Analysis

Since at the point in time this paper was written, the experiment was ongoing work and thus the number of current participants was rather small. Consequently, the data analysed here do not underlie any statistical evidence, but reveal possible trends. Thus, the acceptation or rejection of hypothesis in the following is based on the observed trends.

The experiment’s initial questionnaire showed that both groups were well suited for a comparison. That is due to the quite equal standard of knowledge of feature and decision modelling of all attendees. The questionnaire’s result is shown in Figure 16. The numbers next to the different section are the numbers of attendees for those sections. Even though one half of all attendees exhibited a little more experience with decision modelling then the other half the groups still remain comparable as this divergence is balanced by the initial introduction. Furthermore, none of the attendees knew the Decision Modeller neither in its plain version nor in the extended version.

Figure 16: Modelling Experience

Figure 17 shows the average time needed by both groups to accomplish the different tasks. The tasks are shown on the x-axis, whereas, the average time needed by the attendees is shown on the y-axis. For each task the average of both groups is contrasted by two columns.

Figure 17: Comparison of the average time behaviour

Figure 18 shows how faulty a solution is. A 0 (on the y-axis) indicates a solution without any faults, whereas, a 3 indicates an insufficient solution with too many faults. Again, for each task the average of both groups is contrasted by two columns.

Figure 18: Comparison of fault behaviour

After each task the attendees were asked for their subjective sensation about the task’s difficulty. The results are depicted in Figure 19. The y-axis is shows values between 1 and 4. 1 indicates a very easy sensation and a 4 a very hard sensation. Like in the charts before the results for each group are shown in separate columns.

Figure 19: Subjective sensation of difficulty
Hypotheses H1 & H2

Since complexity and scalability are properties which influence each other they could not be dealt with separately. Consequently, hypotheses H1 and H2 were both addressed by the main task 1 and 2 and its sub-tasks. The tasks demanded the subjects to model constraints among decisions and to extract certain information from a rather small but medium complex decision model which consisted of 15 decisions and 13 constraints. The information to extract was, for instance, related to the model’s integrity. As Figure 17 and Figure 18 show, group2 with the visualization support solved tasks 1 and 2.1 to 2.3 in a shorter time than group2 without the support. Moreover, in almost all tasks group2 made fewer errors than group1 and the task’s post questionnaires yielded that group2 thought that the tasks were easier than group1 thought. Consequently also H1 and H2 could be confirmed.

Hypothesis H3

Task 3 was supposed to evaluate hypothesis H3. Therefore, the experiment’s subjects had to create a possible process which defined in which order the decisions in the decision model should be resolved optimally. Optimally meant that as little decisions as possible had to be made in order to create a resolution model. Since the perfect solution for this task did not exist the subjects’ solution was evaluated with respect to traceability. Thus, it would be optimal to start with decisions referring to requirements and then resolving the according sub-trees which result from constraints among the decisions. Figure 17, Figure 18 and Figure 19 show that no subject from group1 found approximately good solution. Moreover, it took them quite long to accomplish the task and they felt that the task was extraordinarily complicated. To group2, however, the task seemed to be easy. That was also reflected in their solutions. They accomplished the task in a short time and were able to create, not an optimal, but good solution. An optimal solution would have been traversing the process using as few steps as possible. As a result H3 could be accepted.

6. Conclusions

In this paper we proposed several concepts and views on decision models which are supposed to support the management of large-scale decision models. Those were validated in an experiment which indicated that those helped to reduce the complexity and improves scalability and traceability of decision models. Due to time constraints the experiment could only be conducted with a small number of subjects. Consequently, the results can only be seen as evidences without statistical proof. However, the experiment is work in progress and will be executed with further subjects in order to consolidate the statements made here.

7. References