Using Flow Maps to Visualize Product Attributes during Feature Configuration

Abstract—During product configuration one of the main challenges is to understand the available options and the consequences of decisions, e.g., in terms of functionality (features) and other characteristics (attributes) of the product. In this paper, we aim to provide support for understanding consequences of configuration decisions, focusing in particular on numerical product attributes resulting from these decisions. To address this challenge, we introduce “Feature Flow Maps”, an interactive visualization technique, which combines tree-oriented feature models and flow map visualizations. The technique allows the application engineer to simultaneously (1) perform product configuration and (2) to visually understand the resulting numerical product attributes as well as their calculation. We discuss (i) the underlying meta-model including calculation rules, and (ii) the adaptation of flow map visualizations for application in feature models including the mapping of values to visual attributes. 

Keywords—flow maps; feature modeling; product configuration; product line visualization;

I. INTRODUCTION

One of the grand challenges in product line engineering is the complexity caused by the large number of variable (i.e., configurable) elements and dependencies between them. Consequently, during product configuration it is difficult for the application engineer to fully understand the available options and the consequences of decisions.

Examples for consequences of configuration decisions include (1) other configuration options becoming unavailable, (2) satisfaction/violation of constraints, and (3) change in derived attributes of the product.

The indication of other options to become available/unavailable is usually directly supported by configuration tools, e.g., as checkboxes being crossed out if a feature is no longer selectable. Also, many tools provide interaction elements that indicate the violation of constraints, for instance, in the form of list views or markers in the configured model. What is currently insufficient, however, are techniques that provide feedback and show consequences in terms of product attributes.

In this paper, we focus on this third “consequence” of configuration decisions, the resulting attributes of the product. We are particularly interested in numerical attributes, which are calculated as a result of the current configuration. We strive to develop interaction techniques that provide the user sufficient feedback on the values of the product’s (numerical) attributes to support engineers during the configuration process and allow them to understand the consequences of their decisions.

To address this challenge, we introduce “Feature Flow Maps”, an interactive visualization technique, which combines tree-oriented feature models and flow map visualizations.

The technique allows the application engineer to simultaneously (1) perform product configuration and (2) to visually understand the resulting numerical product attributes as well as their calculation. We discuss (i) the underlying meta-model including calculation rules, and (ii) the adaptation of flow map visualizations for application in feature models including the mapping of values to visual attributes.

The technique is implemented and demonstrated using a technical prototype, which extends the S2T2 Configurator for feature models, presented in earlier work. The original version of S2T2 is extended here with (1) support for attributes resulting from configuration, (2) calculation rules on these attributes, and (3) the visualization technique that we call Feature Flow Maps, which are a combination of tree-oriented feature diagrams and flow maps.

II. MODELING CONFIGURED FEATURES AND THEIR NUMERICAL ATTRIBUTES

After introducing the research problem (of providing visual feedback on calculated feature attributes resulting from a feature configuration) we will now take a first step towards a solution by discussing how configured features and their numerical attributes can be modeled.

Imagine a very simple car product line as represented by the feature model in Figure 1. One of the main strengths of feature modeling is the simplicity of feature models, which is achieved by removing all irrelevant information. For instance, a feature model usually does not show technical details on the product’s implementation – and it does so deliberately.

Nevertheless, certain quantitative values concerning the product will be of interest to the user during the process of product configuration. For instance, since every feature of a car has a price this will most likely influence the user’s configuration decisions.
In this context we have to define how the price of a feature can be influenced by the configuration state of other features and how such effects propagate through the hierarchical model. For example if the user selects the Feature Gas from the car example, that decision will influence not only the price of the feature itself (changing it from 0.00 EUR to 5000 EUR) but also the price of its parent feature Engine. This in turn will affect the price of the overall product represented by the root feature Car.

In the context of this paper we will focus on attributes that are (1) numerical and (2) are derived from a given feature configuration (i.e., they cannot be configured directly themselves). We will call such attributes Feature Quantities or short Quantities. Alongside the price other quantities can be defined for a car. Examples are the weight, fuel consumption or the car’s carbon footprint, all of which depend on the configured features.

Because those quantitative structures would provide a valuable guide to the user during product configuration our first step was to create a facility that would allow the valuable guide to the user during product configuration our design and define these models, which we call Quantity Models. Subsequently, we will explain how these quantity models are integrated into techniques for interactive feature configuration by means of the Flow Map Visualization (see Section III).

A. Design Criteria

We start with the requirements that guided the design of the Quantity Model.

1) Multiple Quantity Models might exist for one feature model.

2) The definition of a Quantity Model should require little effort.

3) The definition of Quantity Models should remain flexible. For instance, simple constant values should be possible as well as complex formulas.

4) User errors should be minimized (e.g., by providing validation mechanisms).

5) To support modularization, changes to the core meta-model of the feature modeling approach (in our case fmprimitives) should be avoided.

B. Elements of a Quantity Model

In the following sections Quantity Models and their primary elements are described. Figure 2 shows the elements and their relations in a simple UML diagram.

A Quantity Model is always defined in relation to an existing feature model, to which it adds quantity information in the form of Quantity and Value Computation objects (see the next sections).

Decorator Model Approach: To meet the requirement of avoiding changes to the base meta-model fmprimitives we use a decorator model approach, based on the decorator design pattern [2]. A decorator model enriches an existing base model with additional information while leaving it unmodified. Consequently, this approach enables the definition of multiple Quantity Models for one base feature model, which meets a further design criterion.

Quantities: Quantities are simple entities with a name. They represent the quantitative (numeric) attributes that the Quantity Model provides for the features of the referenced feature model.

Value Computations: In order to describe the value of a feature in matters of a specific Quantity computation rules are provided for the features by the Quantity Model. These Value Computation objects can either simply return predefined constant values or perform more complex calculations like the summation of the quantity values of subfeatures. This way constant feature attributes and complex operations can be handled in a uniform way.

Value Computation usually return a float value. However, if the calculation produced an error or can’t be executed at all (i.e., on division by zero) a predefined error constant is returned. We use Java’s Float.NaN constant for this purpose.

Value Assignments: In order to attach the computation rules to the features, Value Assignment objects are used. These are decorator objects that contain:

- a reference to the feature the Value Assignment adds the information to
- the Quantity for which the calculations produce a value
- a computation rule for each possible configuration state of the feature (Selected, Eliminated, Undecided). A computation has to be defined for at least one of the states.

In a Boolean feature model a feature can be in four different configuration states (undecided, selected, removed, unsatisfiable). Of those, only the first three are relevant in practical feature configuration. Hence the Value Assignment object stores up to three computation rules – one for

1Not a Number

2Either the configuration approach is backtracking-free and avoids unsatisfiable configurations altogether or, whenever the whole product configuration is unsatisfiable, the calculation of the product’s attribute is undefined.
each state. In particular this allows to distinguish between removed and undecided features.

A possible use case in the car example would be the elimination of the Radio feature. The absence of the radio would not result in zero costs because now a cover the for radio slot, which is not modeled as a feature, would be required.

A computation that is directly assigned to a feature by a Value Assignment object is called a local computation. Figure 3 shows an example where features are assigned value computations for two quantities "weight" and "price". Please note that there might features without a Value Assignment (e.g., Feature 3).

Global Computation Rules: Since in practice feature models can grow quite large we decided to simplify the creation of Quantity Models by expecting the user to only assign computations to certain features. Many feature models contain a lot of features that mostly serve as means of hierarchical organization, which can be omitted here. In order to define a computation for every feature with every quantity we introduce global computations: In addition to its name a Quantity object also stores a computation rule for each of the three possible configuration states of a feature (see Figure 2). These stored computation rules apply as default rules. In other words, whenever there is no Value Assignment for a features, which would provide a computation rule for its current configuration state and the currently calculated Quantity, the correspondent global computation rule is applied. Local rules always override global rules. In contrast to the Value Assignments it is mandatory to define a rule for all three states. In doing so we ensure that at least a global computation rule exists for each feature and each Quantity.

Types of Value Computations: The following list describes different types of Value Computations available in the Feature Quantities meta-model:

- **Constant Value** is the most basic Value Computation.
It just returns a predefined value.

- **Value Computations for sets of features**
  The following operations are available for sets of features: sum, product, average, minimum and maximum.

There are two different classes of computations that work on different feature sets:

- **Feature computations** work on a specific feature set defined by the creator of the *Quantity Model*. They mostly serve as local computations to modify the behavior imposed by the global computation rules.

- **Subfeatures computations** operate on the subfeatures of the feature they are applied to. The model creator doesn’t need to specify the target feature set because it can be inferred. Subfeature computations also serve well as *global computation rules* because the operand feature set is looked up every time the calculation is invoked. Additionally a predefined value for empty subfeature sets can be defined. Therefore they work on all features in the model.

- **Composite computations**
  The operands of these computations aren’t features but other *Value Computations*. With basic operation like sum and product this enables the construction of more complex formulas.

  With the presented model elements, a *Quantity Model* can calculate a numeric quantity value for every feature of the base feature model. The next section describes the integration of these values into the feature model visualization.

### III. Visualization

This section describes the Flow Map visualization as a means to integrate the numerical feature attributes calculated by the *Quantity Models* into the interactive configuration environment of the *S2T2 Configurator* research prototype. We begin with the design criteria that guided us during the development of the visualization. Afterwards, we introduce Flow Maps in general and then describe our adaption to feature models. Finally we present and discuss the result.

#### A. Objectives for the Visualization

This section lists our objectives for the development of the Flow Map visualization. The first five items describe more general and partly competing objectives regarding Information Visualization [3]. The further items are specific to our project.

- **Effectiveness**
  A visualization is regarded as *effective* if the information to be conveyed is perceived intuitively and without increased cognitive effort.

- **Expressiveness**
  Expressiveness will be achieved if the user grasps the information (and only those) contained within the presented data set and gains a better understanding of said data.

- **Avoiding Information Overload**
  In many cases it is possible to encode a lot of information in a visualization. Mostly though, this is not reasonable because it may lead to Information Overload so that the user can not concentrate on the essential.

- **Avoiding Clutter**
  Clutter [4] can arise if too many or to large graphical elements are visible or are arranged or rendered in an unfavorable way. Information Overload and Clutter often appear together if too many artefacts or graphically encoded attributes are presented.

- **Exploiting the available screen space**
  The limited available screen space should be utilized effectively. This criterion might compete with the criteria for avoiding Clutter and Information Overload.

- **Integration into the S2T2 Configurator**
  The developed quantity visualization should be integrated with the tree visualization of the *S2T2 Configurator* by extending and complementing it.

- **On-demand support**
  It should be possible to switch the quantity visualization on and off without compromising the overall display.

- **Preserving existing visual elements**
  Information presented by the existing visualization should not be obscured or hidden.

#### B. Flow Maps

Flow Maps have their roots in geography, where they have been used for a long time to illustrate the movement of objects between different locations in conjunction with their number or some other quantitative dimension. Examples are the migration pattern of birds or the movements of goods. Figure 4 shows a historic example by the French cartographer Charles Joseph Minard depicting the wine exports of France in 1864.

#### C. Adaptation to Feature Models

Flow Maps are suitable for the interactive visualization of quantities within the *S2T2 Configurator* because they can be combined with the tools’ tree-like graph presentation. Feature models are organized hierarchically in many cases so the quantity values of child features will be accumulated or selected by their parent feature. The values proverbially “flow” from the leaves to the root.

By visually encoding the values from the *Quantity Model* into the feature nodes and hierarchy edges (subfeature- and feature group-relations) the existing tree view and the visualization of quantities can be combined.

As the preferred visual attribute for quantitative values we chose the spatial dimension of the visual entities. In the case of the *S2T2 Configurator* these are the size of the...
feature nodes and the thickness of the edges. Figure 5 shows a concept draft of the Flow Map visualization.

In order to achieve a visually “homogeneous” flow the thickness of the edges and the dimension of the feature nodes should be equal. Features that have a value of 0.0 or the error value Float.NaN must nevertheless still be visible. Therefore only the thickness of the hierarchy edges changes consistently over the whole domain of the quantity values. The adjustment of the size of feature nodes is only performed for values above a certain base value. See Section III-E for further details.

D. Hierarchy Edges

Every hierarchy edge is represented by a base line with constant thickness. Edges with a positive quantity value (not equal to 0.0 or Float.NaN) are additionally rendered with a colored hull. The spatial extend of that hull is adjusted according to the displayed value. To make sure that base line and hull can be distinguished, the base line is drawn in a dashed line style while the hull is solid. The base line of edges with the value Float.NaN are additionally rendered in red to signal their error state.

The concrete value which is displayed by an edge depends on its adjacent entities. Subfeature edges display the quantity value of the subfeature, so the accumulated thickness of all subfeature edges will comply with the dimension of the parent feature node. In Figure 5 these are the edges that go from feature A to its subfeatures. Edges going from features (G and H) to a superior feature group are handled in the same fashion. However, the edge between a feature and its inferior feature group (feature B an its feature group) shows the value of the feature, because the value can not change between both objects.

E. Feature nodes

The S2T2 Configurator displays feature nodes as shown in Figure 6 with the name of the feature, a configuration field and a button for expansion control where necessary. Feature group nodes are displayed similarly but with the cardinality instead of the name and a FODA-notation inspired icon [5] instead of the configuration symbol.

Because the display of feature- and feature group nodes is very similar we will concentrate on the feature node in the following sections.

1) Size scaling: In order to integrate the Flow Maps harmonically into the existing feature model visualization, the display of the feature nodes should change as little as possible but must respond to size changes at the same time. Simply scaling the feature nodes seems to be intuitive at first glance but turns out to be problematic at a closer look. The reason for this will be discussed in the next sections:

Font size: To begin with, the scaling of the label text in the feature node leads to an inhomogeneous size perception. A text that mainly consists of narrow letters (i.e., “i”, “l” etc.) often appears smaller than a text with many wide letters. Figure 7 shows an example for this effect. The labels of rectangle (a) and (b) seem to be formatted with the same font size. But the font size in rectangle (b) is in fact two points smaller than in (a).

Furthermore, text attracts much more attention than simple shapes. As Figure 7 demonstrates, the human perception of text size is quite inaccurate. Since our focus lies on the
visualization of quantitative structures, using the font size as a means of illustrating these values would therefore be inadvisable.

UI Controls: When using a simple scaling of the feature nodes, the expansion control button and the configuration field would be displayed in different sizes for different quantity values. This would lead to a visually homogeneous user interface which could confuse the user.

Node hull: For the named reasons we waived the scaling of the entire feature node. Instead the node is displayed in the usual manner but encompassed with a hull in the shape of a filled rectangle. The hulls’ dimensions are computed from the features’ quantitative value. Figure 8 shows a feature node in the undecided state without a hull and another one in the selected state with a hull that represents a quantity value of 5000.0.

As we have already explained, the feature nodes themselves should not be scaled in order to remain recognizable at all times and avoid confusing the user. For the same reason they will never be shrunk and so the hull is only visible if its dimension is bigger than the dimension of the feature node. This is determined by the method of mapping quantity values to visual size, which is described in Section III-F.

Here, all that should be mentioned is that the visual size of a feature node has always a value of at least 1.0. A hull is only drawn if the visual size is greater than that base value. If it is equal to or lower than 1.0, the user can still recognize the quantity value by the size of the hierarchy edge-hull, where no such adjustment is necessary. Figure 9 shows such a case.

F. Mapping quantity values to visual size

The objects in the visualization of the S2T2 Configurator are drawn by so called renderers. What objects they draw and how their display is changed by filters or other manipulations of the visualization the renderer finds out by a set of properties that are stored in the visual objects. Examples for said properties are the position, fill- or line color, visibility etc. Another important attribute is the visual size, a scaling factor represented as a floating point value. If an object should displayed in its original size, the visual size is 1.0. This value shall be called the visual base size. If and how this parameter affects the presentation of an object is decided by the renderer that draws the object.

When mapping quantity values to visual attributes the visual size of a feature node (or an edge) is a suitable choice. Several options for the mapping are available:

- **Maximum Mapping** For this mapping the currently highest quantity value $m$ in the model is found and the interval $[0.0, m]$ is mapped to the visual size interval $[0.0, 10.0]$ linearly.
  
  Pros: For this mapping method the user himself doesn’t need to set reference values. Furthermore, objects with a visual size of more than 10.0 are reasonable only to some extent because they tend to cause clutter.
  
  Cons: Every configuration decision changes the maximum quantity value (which mostly is computed at the root feature). This causes a “bumpy” visualization because the continuous change of the mapping reference point also causes the continuous change of the visual size of all objects.

- **Base Value Mapping** The quantity value that is mapped to the visual size of 1.0 is defined explicitly by the user. From there the mapping for every other value can be inferred.
  
  Pros: The visualization remains stable on configuration decisions because of the fixed reference value.
  
  Cons: The suitable base size value will most likely differ from model to model. So the user has to adjust this value manually. Also heuristics would be imaginable but do not guarantee reasonable results.

- **Logarithmic Mapping** This mapping can be combined with one of the aforementioned mapping methods. It is useful if many features with a median value and a small number of features with large values exist in the visualization (similar to a normal distribution). The logarithmic mapping prevents the few large values from dominating the many median values.
G. Result and Discussion

The Flow Map visualization provides an intuitive overview of the quantitative structures within the feature model while abstracting from unnecessary details. The existing feature model visualization of the S2T2 Configurator is enriched with numerical information without hiding or obscuring the existing elements.

Early experiments with the prototype indicate that the approach can be considered useful since it helps the user comprehend the consequences of his configuration decisions easier and more quickly. When dealing with prices of the configured product parts it is easy to see that the spatial Flow Map representation of those cost data can be comprehended with less cognitive effort than a display of mere numbers.

Furthermore, the approach allows the user to quickly orientate in the model and gain an overview of the cost drivers within the product. This can be done by simply looking at the visualization and identifying the biggest flows. The effectiveness criterion can therefore be considered satisfied.

At all times, the user can recognize which portion specific features hold at the overall cost of the product. For this reason, the visualization is also expressive to some extent.
As an example consider Figure III-E1 which shows the visualization of the example car feature model. In order to avoid information overload, we decided to omit the display of precise values. Such precise values could be displayed on demand, but would cause visual clutter if shown permanently.

For near-by edges with great thickness clutter can occur because of the used Bézier curves as depicted in Figure 11. One possible solution would be to increase the horizontal offset between the nodes which however would decrease the exploit of the visible screen space. Since the user can move the nodes freely within the S2T2 Configurator he can correct the display manually if necessary.

Furthermore, we want to note that the employment of primary colors for the edge hulls can cause other elements to be less noticeable. Therefore, the hierarchy edges should be drawn in a more unobtrusive color. We however have chosen a spot color for a better contrast and readability in this article.

In both small and bigger feature models the Flow Map visualization works quite well as Figure 12 demonstrates.

The mentioned statements, e.g., on usefulness and fitness for the focussed tasks of product configuration involving quantitative information, are based on early informal experiments made by the authors. In future work these need to be substantiated by a more formal evaluation.

IV. RELATED WORK

The visualization of hierarchical structures has been studied extensively in the literature, including node-link techniques (e.g., [6], [7]) and space-filling techniques (e.g., Tree Maps [8], [9]). The research described in this paper strives to support product configuration by combining such hierarchical structures (i.e., the feature model) with quantitative information (e.g., the price of a configured product) and visualizing both, the hierarchy and the quantitative information, together.

Similar techniques can be found in On-Line Analytical Processing (OLAP). For instance, Mansmann and Scholl [10] describe hierarchical visualization techniques for OLAP aggregates. The Polaris system [11] extends the interaction technique of Pivot Tables to allow for a visual query of multidimensional databases. Similar to the expand/collapse interactions in feature models, such interactive OLAP approaches support drill-down operations. They differ from feature models in the sense that they aggregate along multiple dimensions of a data cube (e.g., regions, time intervals, product types) and each drill-down operation can occur along a dimension of the user’s choice. Another difference lies in the fact that, an OLAP tool only allows the user analyze a fixed data set, but does not provide configuration operations, which change the underlying data set.

There is a large number of research prototypes which provide feature modeling and configuration capabilities. Examples for commercial configuration tools in the product line context are pure:variants [12] and BigLever’s Gears [13].

Rabiser et al. [14] describe a wizard-oriented approach to product configuration, where decisions that are currently not available are filtered out. Sellier and Mannion [15] present their V-Visualize tool which visualizes configuration decisions with a force directed layout.

The Prefuse visualization framework, which we used as a foundation for the approach presented here, is described in more detail by Heer et al. in [16].

In earlier work we presented Visit-FC, a configuration approach that indicates the configuration state of feature nodes by visual clues [17], [18]. More earlier work includes the software design of S2T2 Configurator which integrates formal foundations and interaction techniques [1], techniques for the configuration of complex feature models [19], and techniques for the joint visualization of feature and implementation models [20].

V. CONCLUSIONS

We have introduced Feature Flow Maps an interactive visualization to be used during feature configuration. The technique provides additional information on the consequences of configuration decisions. In particular, it shows attribute values that develop while a partial configuration is completed with more and more configuration decisions. For instance, a user can see how the price of the product increases and decreases while he adds functionality or revises his decisions because the price is exceeding his budget. Besides the information on the changing attribute value itself, the user also is provided with visual clues on the composition/calculation of this value.

Of course, the presented approach also has its limitations. The approach (including the configuration semantics and the capabilities of the applied reasoning engine), supports Boolean feature configuration (i.e., features can be selected and eliminated) and the calculation of resulting numerical attributes. It does not support the configuration of numerical attributes by the user. For instance, the user can select a feature “Diesel engine”, which results in the weight of the car increasing by 200kg, but he/she cannot set the weight of the car to 1500kg as a direct configuration decision. Also, in the current state of the approach, the calculated attribute values are not used within constraints, such as weight(body) + 4 * weight(wheel) < 2000kg.

Future work includes the validation with real industry cases, evaluation of the scalability in terms of performance and interaction complexity as well as the extension of the modeling approach towards more expressive languages and reasoning engines.
Figure 12. Larger feature model with Flow Map visualization (overview and zoom)
ACKNOWLEDGMENT

This work was partly supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero, http://www.lero.ie/.

REFERENCES


