Interactive Techniques to Support the Configuration of Complex Feature Models

Goetz Botterweck[1] and Denny Schneeweiss[2] and Andreas Pleuss[1]

[1] Lero, University of Limerick, Ireland,
{goetz.botterweck|andreas.pleuss}@lero.ie
[2] BTU Cottbus, Germany,
denny.schneeweiss@tu-cottbus.de

Abstract. Whenever a software engineer derives a product from a product line, he has to resolve variability by making configuration decisions. This configuration process can become rather complex because of dependencies within the variability model and knock-on effects and dependencies in other related artefacts. Because of the limited cognitive capacity of the human engineer, this complexity limits the ability of handling product lines with large configuration spaces. To address this problem we focus on techniques that support the interactive configuration of larger feature models, including (1) visual interaction with a formal reasoning engine, (2) visual representation of multiple interrelated hierarchies, (3) indicators for configuration progress and (4) filtering of visible nodes. The concepts are demonstrated within S^2T^2 Configurator, an interactive feature configuration tool. The techniques are discussed and evaluated with feature models, however, we believe they can be generalised to other models that describe configuration choices, e.g., variability models and decision models.

1 Introduction

In Software Product Line (SPL) Engineering we are dealing with interrelated, complex models. A common concept for modelling a product line are feature models. They are usually interpreted as a tree structure consisting of feature-subfeature and group-member relations. In addition, there are other relationships across the hierarchy and across different levels (e.g., *requires*, *mutex*). The most common visual representation for such models is a graph [1,2].

When deriving products from a product line, the software engineer has to resolve variability by making configuration decisions. This configuration process and the decisions can become rather complex because of the dependencies within the variability model and knock-on effects and dependencies in other related artefacts that represent the product line. Because of the limited cognitive capacity of the human engineer, this complexity limits the ability of handling product lines with large and complex configuration spaces.

Hence, there is a need for tool support for product configuration. Existing commercial feature modelling software like pure::variants [3] provide basic

graph-based visualisations of feature models. Heidenreich et al. [4] introduce FeatureMapper, which supports mapping features to model elements in arbitrary (EMF-based) models. The mappings are expressed by colouring the elements corresponding to each feature. However, there is still a lack of more advanced visual and interactive tool support for handling product derivation.

We address this problem with S^2T^2 *Configurator* a feature configuration tool which integrates an interactive visual representation of the feature model with a formal reasoning engine, that calculates consequences of the user's decisions and provides formal explanations.

In addition, we support to visualise the relationships between selected features and further aspects of the software product line, like a component model specifying the architecture. This requires to present multiple hierarchies (e.g., features and components) as well as the relationships between them, e.g., which features are implemented by which components.

In earlier work [5] we discussed the visualisation of product lines. In [6] we introduced the software architecture for our Configurator tool, designed as a chain of configurator components, to allow flexible integration of different models and a formal reasoning engine. In this paper we focus on interaction techniques which support the interactive configuration of larger feature models, including the visual interaction with the reasoning engine (section 3.1), visual representation of multiple interrelated feature trees (section 3.2), indicators for configuration progress (section 3.3) and filtering (section 3.4). All described techniques have been implemented in a prototype which is available for download[3].

2 Analysis of Possible Solutions

To address the configuration of complex feature models, we evaluated visualisation techniques with respect to their suitability for typical structural characteristics of such models. Feature models are usually structured as hierarchies, made up of feature-subfeature and group-member relations [1,2]. In addition, depending on the type of feature model, there can also be arbitrary relationships between models elements, independent of the main hierarchy (cross-tree constraints or *intra-model relations*). Further challenges are given by large models (not fitting on the canvas) and multiple, interrelated models with relationships between them (*inter-model relations*).

We classify the evaluated approaches into techniques for visualisation of tree hierarchies (section 2.1), interaction techniques for large data structures (section 2.2), and approaches focussing on multiple hierarchies (section 2.3).

2.1 Visualisation of Tree Hierarchies

The area of Information Visualisation provides various alternative approaches to visualise tree hierarchies:

[3] http://download.lero.ie/spl/s2t2/

Space-filling visualisations such as tree-based maps [7] or Sunbursts [8] provide an insight into the quantitative relations within a hierarchy. Although these types of visualisations are quite effective in this respect, feature model visualisations in general do not benefit from it. It is difficult to enrich them with additional information (i.e., text rendering in small cells is problematic) and displaying relations between elements leads in almost all cases to visual clutter. This problem gets even worse with multiple hierarchies.

Three-dimensional visualisations such as Cone Trees [9] are problematic when applied in interactive configurations since certain elements may be hidden by others. Moreover the navigation and orientation within a 3D environment is challenging for most users.

In general, various user tests on tree visualisations have shown that mature common tree visualisation systems (like file explorers) often perform as good or even better as alternative visualisations (e.g., [10]). Thus, it seems reasonable for feature model visualisation to decide for a conventional 2D tree visualisation and put effort on its optimisation and enhancement. As a first step, techniques from Graph Drawing can be used to reduce the graph's complexity, e.g., by minimising intersections, providing a well-structured spatial layout or reducing the required area.

2.2 Interaction Techniques for Large Data Structures

The second important area arises from the interactive nature of the Configurator tool. Certain interaction techniques enable the user to focus on the information in the centre of his or her interest while muting currently unimportant information. Common techniques in these area are for instance [11]:

The ***overview+detail*** interface design, which is characterised by the use of both an overview and a detail view of the same information space, in *separate* presentation spaces (spatial separation of focus and detail).

focus+context interfaces integrate both details and overview seamlessly into a single display (spatial separation, but with a seamless integration).

A ***zooming*** interface design provides access to overview and detail in temporal separation. Here the users first zoom out to get an overview and then zoom in to see the details.

These are promising approaches and are applied in our tool. To identify which elements are part of the user's focus and which elements are outside of the scope of interest at one point of time it is necessary to consider domain-specific knowledge and the semantics of the models.

2.3 Approaches for Visualising Multiple Hierarchies

Approaches focussing on multiple hierarchies are useful when visualising the relationships between features and other models as explained above. Robertson et al. define polyarchies [12], multiple hierarchies that share nodes. They describe the visualisation design and a system architecture for displaying polyarchy data

from a set of hierarchical databases. They use various animated transitions when switching between the related hierarchies to allow the user to keep context. Polyarchies are somewhat similar our multiple related hierarchies, but lack the intra-model relations and the aspect of progressing configuration.

In later work the authors address visual mappings between two hierarchical schemas [13]. They use auto-scrolling and focusing-techniques to display distant related elements within the boundaries of the screen. Again the data structures are different from feature models (no intra-model relations etc.); nevertheless some of the interaction techniques could be adapted for feature models.

3 Interactive Techniques in the S^2T^2 Configurator

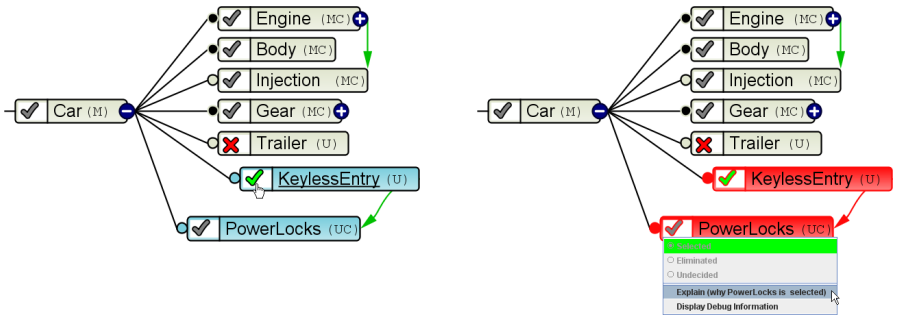
As argued above, we chose a two-dimensional tree hierarchy as the basic visualisation. The configurator supports all basic interaction techniques for tree structures like *collapse/expand functionality*, *panning*, and *seamless zooming*. The following sections describe the more advanced features in greater detail.

3.1 Interaction with Formal Semantics

The S^2T^2 Configurator integrates a reasoning engine that supports interactive functionality such as calculating the consequences of his or her decisions based on the formal semantics of the models [6]. We interpret a feature model as (1) a set of features and (2) a set of constraints over these features. Moreover, (3) we interpret features as variables with value domains limited by the given constraints. For instance, in boolean feature models, there are four potential configuration states: *Undecided* $\{true, false\}$, *Selected* $\{true\}$, *Eliminated* $\{false\}$, and *Unsatisfiable* $\{\}$. During the configuration process, each user decision–and the consequences calculated from it–are interpreted as additional constraints further reducing the available values in the domains. When the configuration process is completed and all variability has been resolved, there is exactly one possible value left for each feature. Similar concepts can be defined for non-boolean feature models by allowing larger value domains (e.g., to specify the maximum price for a car).

Figure 1 shows screenshots from our tool with a small example feature model. The basic visualisation follows common tree-based feature model representations. Additionally, a symbol represents the feature's state: a check mark indicates that the feature is selected, while a cross indicates that the feature is eliminated. The user can modify the feature's state by clicking on this area. Symbols in grey colour indicate that the state is already given by constraints and cannot be changed.

When the model is initially loaded and after each user decision, the reasoning engine automatically calculates the consequences and applies them to the model. In figure 1(a) the Configurator inferred that `Injection` has to be selected as it is required by the mandatory `Engine`. The screenshot shows the moment where the user sets the feature `KeylessEntry` as selected. As it requires the feature



(a) Calculated consequences. (b) Explanations for consequences.

Fig. 1. Configuration with consequences and explanations.

`PowerLocks`, the Configurator automatically sets `PowerLocks` as selected as well. Whenever the Configurator automatically applies changes, this is indicated with an animated blue decoration.

The annotations on the right-hand side in each node denote whether a feature's state has been set by the user (U), is a consequence from a user decision (UC) or is specified by the model (M).

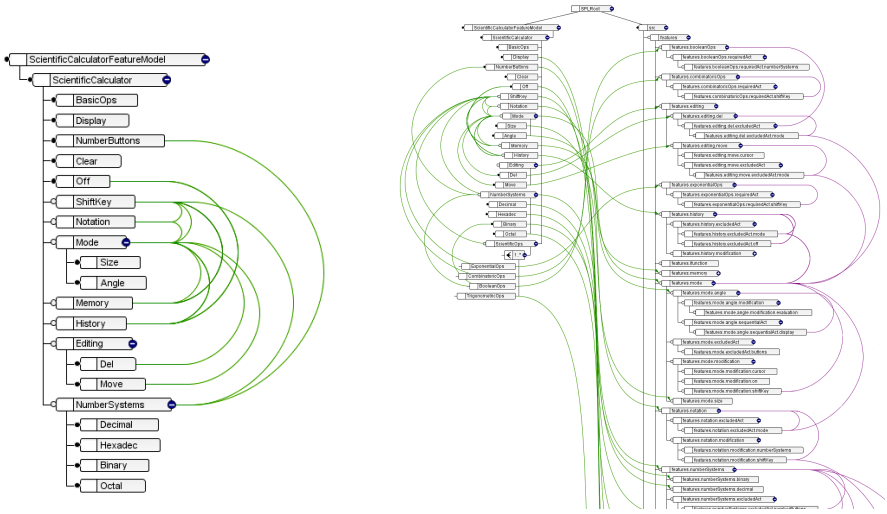
Although in such simple examples the reasons for a consequence are obvious, in more complex feature models the user might not immediately comprehend the situation. Hence, the Configurator supports visual explanations, which are based on formal explanations calculated by the reasoning engine. This enables the user to ask for an explanation (using the context menu) why a certain feature is selected/eliminated. For instance, in figure 1(b) when the user asks why `PowerLocks` is selected automatically, the Configurator will respond by highlighting `KeylessEntry` and the `Requires` edge between `KeylessEntry` and `PowerLocks`.

3.2 Multiple Related Models

Due to its modular design described in [6], S^2T^2 Configurator supports the seamless integration of different kinds of models. As discussed earlier, one goal is to visualise multiple interrelated hierarchies while supporting the user's orientation within the complex model, e.g., by avoiding intersections of edges and similar visual clutter.

The Configurator supports different tree-layouts, e.g., a vertical tree layout with indentations, see figure 2(a). Hierarchy edges are rendered rectilinear while additional relations are shown as semicircles with adjusted diameters. Thus, we avoid crossing feature with edges and two edges only intersect if their nodes are interleaved. Moreover, edges that start and end in similar positions can be easily distinguished in most cases since they usually differ in their horizontal extend.

For the visualisation of two interrelated models we adapted this approach by positioning the trees side-by-side and changing the orientation of the left model



(a) Vertical Tree Layout with semicircles.

(b) Multiple feature hierarchies and their relationships

Fig. 2. Approaches to avoid clutter in hierarchies with cross-model-relations

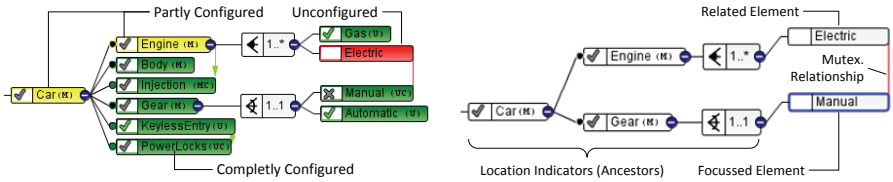
to be rendered with right-aligned nodes, see figure 2(b). Intra-model relations are rendered as semicircles on the outer side of each model, inter-model relations as cubic splines connecting both models. With this layout variation the clutter is kept in reasonable bounds even with two interrelated models.

Our tool supports displaying more than two hierarchies, but then it becomes hard to avoid intersections. Hence, we recommend to avoid such scenarios by letting the user interactively chose two models to be displayed on the left- and right-hand side; similar to common file manager tools like Norton Commander.

3.3 Progress Indicators

With progress indicators we aim to distinguish areas that have been configured from areas that still need attention. A feature node is *configured* if all decisions with respect to this feature have been made. In general, this is the case if the value domain has been reduced to one element; for boolean feature models this means that the feature is either *Selected* or *Eliminated*.

If we aggregate the configuration state of all nodes in a subtree, we can distinguish between three levels of progress for the subtree, *Unconfigured*, *Partly configured*, and *Completely configured*. If we apply results from research on colours and selective attention, we can colour the root of each subtree accordingly and draw the user's attention to those locations where decisions are pending. When the configuration process progresses these colours change and become less emergent. Figure 3(a) shows this in a simple feature model.



(a) Progress indicators. (b) Filtering.

Fig. 3. Progress indicators and filtering.

3.4 Filtering

Another technique to improve the interaction with complex models is to (1) differentiate between elements that are *currently* relevant ("in focus") and those that are not and (2) use this categorisation in appropriate interaction techniques, e.g., by filtering out non-relevant information.

To find relevant nodes, we partitioned the model elements into four sets: *Focussed elements*, *Related elements* (linked directly or indirectly to focussed elements), *Location indicators* (necessary to understand the logical position within the overall model, e.g., all ancestors of focussed elements), and *Other elements*.

Whenever the user focusses on one feature (double click) the Configurator calculates the (directly) related elements. The user can then hide all other elements to concentrate on the current configuration decision (see figure 3(b)). We also display the ancestors of the focussed element to show the relative position within the model.

4 Conclusions and Future Work

In this paper we have discussed techniques that support the interaction with and configuration of complex feature models. The techniques have been demonstrated with the research prototype S^2T^2 Configurator. A beta version of the prototype is available at `http://download.lero.ie/spl/s2t2/`.

After the initial design and realisation of the discussed techniques we gained more insights by experimenting with three test cases: (1) Generated feature models with varying sizes and distributions of element types (2) a product line of parking assistant applications [14] and (3) a calculator case study [15], which contains a feature model and a model of the implementation as well as feature-implementation mappings. In general, this allowed us to perform a first internal evaluation of our approach. In summary, the introduced techniques seemed to improve the situation with respect to the size of models that can reasonably be handled. As next steps we plan to perform user tests and provide a formal evaluation to substantiate this with empirical evidence. Based on that experience we intend to extend the Configurator, e.g., defining additional filtering rules based on common tasks in product-line engineering.

Acknowledgments This work was partly supported by Science Foundation Ireland grant 03/CE2/I303_1 to Lero – The Irish Software Engineering Research Centre (http://www.lero.ie/).

References

1. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature oriented domain analysis (FODA) feasibility study. SEI Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute (1990)
2. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software Process Improvement and Practice **10**(1) (2005) 7–29
3. Beuche, D.: Variants and variability management with pure::variants. In: 3rd Software Product Line Conference (SPLC 2004), Workshop on Software Variability Management for Product Derivation, Boston, MA (August 2004)
4. Heidenreich, F., Kopcsek, J., Wende, C.: Featuremapper: Mapping features to models. In: ICSE Companion '08. (2008) 943–944
5. Botterweck, G., Thiel, S., Nestor, D., bin Abid, S., Cawley, C.: Visual tool support for configuring and understanding software product lines. In: SPLC 2008, Limerick, Ireland (September 2008)
6. Botterweck, G., Janota, M., Schneeweiss, D.: A design of a configurable feature model configurator. In: VAMOS 2009. (2009)
7. Johnson, B., Shneiderman, B.: Tree-maps: A space-filling approach to the visualisation of hierarchical information structures. IEEE Visualization (October 1991) 189–194
8. Stasko, J., Zhang, E.: Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In: Proc. of IEEE Information Visualization 2000, Salt Lake City, UT (2000) 57–65
9. Robertson, G.G., Mackinlay, J.D., Card, S.K.: Cone trees: animated 3d visualizations of hierarchical information. In: CHI '91. (1991) 189–194
10. Kobsa, A.: User experiments with tree visualization systems. In: INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization, Washington, DC, USA, IEEE Computer Society (2004) 9–16
11. Cockburn, A., Karlson, A., Bederson, B.B.: A review of overview+detail, zooming, and focus+context interfaces. ACM Comput. Surv. **41**(1) (2008) 1–31
12. Robertson, G., Cameron, K., Czerwinski, M., Robbins, D.: Polyarchy visualization: Visualizing multiple intersecting hierarchies. In: CHI'02. Visualizing Patterns, ACM Press (2002) 423 – 430
13. Robertson, G.G., Czerwinski, M.P., Churchill, J.E.: Visualization of mappings between schemas. In: CHI '05, New York, NY, USA, ACM (2005) 431–439
14. Polzer, A., Kowalewski, S., Botterweck, G.: Applying software product line techniques in model-based embedded systems engineering. In: MOMPES 2009, Workshop at ICSE 2009, Vancouver, Canada (May 2009)
15. Lee, K., Botterweck, G., Thiel, S.: Aspectual separation of feature dependencies for flexible feature composition. In: COMPSAC 2009, Seattle, WA (July 2009)