Visual Configuration in Automotive Software Product Lines

Goetz Botterweck[1], Steffen Thiel[1], Ciarán Cawley[1], Daren Nestor[1], André Preußner[2]

[1]*Lero,University of Limerick*
Limerick, Ireland
{ goetz.botterweck | steffen.thiel |
ciaran.cawley | daren.nestor }@lero.ie

[2]*BTU Cottbus*
Institute of Computer Science
Cottbus, Germany
apreussn@informatik.tu-cottbus.de

Abstract

Software Product Line engineering has emerged as a viable and important software development paradigm in the automotive industry. It allows companies to realise significant improvements in time-to-market, cost, productivity, and system quality. One major difficulty with software product line engineering is related to the fact that a product line of industrial size can easily incorporate thousands of variation points. This scale of variability can become extremely complex to manage resulting in a product configuration process that bears significant costs. This paper introduces a meta-model and research tool that employs visualisation and interaction techniques to improve product configuration in high-variability product lines.

1. Introduction

Software Product Line (SPL) engineering is a paradigm to develop software-intensive products using platforms and mass customisation. This is achieved through the identification and control of the products' commonality and variation. Adopting a product line approach allows companies to build a variety of systems with a minimum of technical diversity and to realise significant improvements in time-to-market, cost, productivity and quality [1]. The effective management of a product line's variability is key to its success. Particularly in the area of feature modelling and product configuration, variability management can greatly impact the complexity that is involved when developing new products from existing product line assets [2].

Within the automotive industry, product lines of automotive systems exist with thousands of variation points and configuration parameters that need to be managed in order to customise a particular product [3]. Managing this level of variability is extremely complex and can be very costly [4]. Furthermore, in cases such as these where there are a large number of variants, appropriate techniques are required to allow particular stakeholders to perform their specific tasks [5].

One technique that can be applied beneficially in this context is visualisation. Visualisation takes abstract data and transforms it into a format that is useful for presentation to humans. In doing this, human cognition is enhanced and understanding is afforded. In the area of software product line variability management, visualisation can be used to amplify cognition of the large, complex data sets that can exist in industrial SPL engineering.

This paper presents a meta-model and a prototype tool for feature configuration. The tool implements various visualisation and interaction techniques that can support stakeholders in the process of product configuration for high-variability product lines.

The remainder of this paper is organised as follows: in Section 2 we summarize a meta-model for feature configuration in software product lines. In Section 3 we introduce our visual research tool (VISIT-FC) which is based on the meta-model and explain some of the visualisation and interaction techniques implemented. Section 4 provides an illustrating example of a feature configuration for an automotive restraint system using VISIT-FC. Section 5 discusses related work in visual product configuration and Section 6 outlines future work. Finally, Section 7 concludes the paper.

2. Feature Modelling

A key aspect of the SPL engineering approach is the modelling of the variability of the supported product line features. Such a feature model allows for inclusion and exclusion of various features and variants so that a valid feature configuration is produced. A feature model also guides product configuration and can be used to validate a particular configuration for conformance.

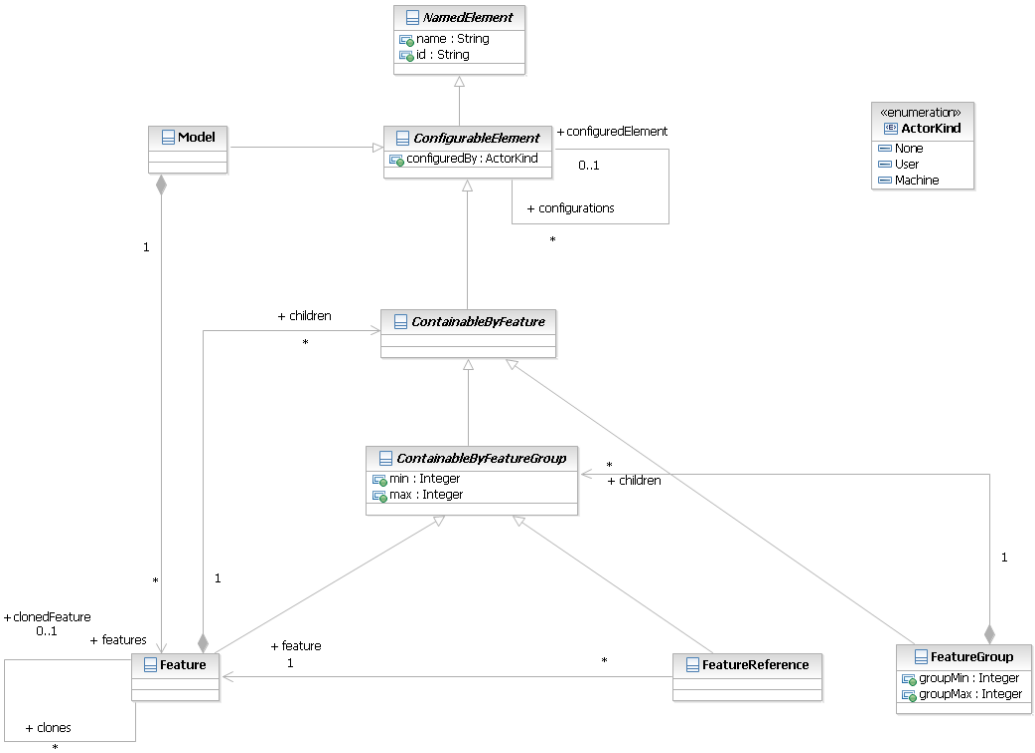

Figure 1. Overview of the Feature Model

Figure 1 provides an overview of a meta-model which we developed to describe software product line feature models. The model is an extension of Czarnecki's work [6] and forms the basis of a prototype tool presented in sections 3, 4 and 5. In the following sub-sections we introduce the main characteristics of this meta-model.

2.1. Basic Model Structure

The model structure is designed to support a staged configuration approach where a model may be loaded, partially configured/constrained and saved in iterations. This allows a product to be gradually configured with each stage extending on the previous until all feature variability has been resolved and an end product has been configured. This is supported through the sub-classing of `ConfigurableElement` which can contain many `configurations` and can itself be a configuration of one `configuredElement`.

The model supports a hierarchy of features and feature groups where a `Feature` can contain `Features`, `FeatureGroups` and `FeatureReferences`. By using a generalisation / specialisation association between `ContainableByFeature` and `FeatureGroup` and a composition association between `FeatureGroup` and `ContainableByFeatureGroup` we enforce that a `FeatureGroup` cannot contain other `FeatureGroups` but can contain `Features` and `FeatureReferences`.

2.2. Cardinalities

Element selection and elimination is modelled using cardinalities. A `Feature` and `FeatureReference` have a minimum and maximum denoting the number of times they occur [min, max]. This allows us to model optional features as [0, 1], mandatory features as [1, 1] and eliminated features as [0, 0].

A `FeatureGroup` has a `groupMin` and a `groupMax` attribute denoting the minimum and maximum number of elements that can be contained within them. As an example, a `FeatureGroup` containing a set of alternative features would be modelled as `groupMin=1`, `groupMax=1` and each `Feature` within the `FeatureGroup` would have their `min` and `max` atributes set to [0, 1].

2.3. Dependencies

The meta-model supports two types of feature relationships, an `UndirectedDependancy` and a `DirectedDependancy` (not shown in Figure 1). Two concrete implementations of a `DirectedDependancy` are `Requires` and `Recommends`. As the names suggest, a `Requires` dependency denotes that if a source

feature is selected then the target feature *must* also be selected. A `Recommends` dependency denotes that if the source feature is selected then the target feature *should* also be selected.

Supported implementations of an `UnDirectedDependancy` include `MutualExclusion` and `MutualProblematic`. `MutualExclusion` denotes that if any one of the set of features is selected then the other feature(s) *must not* be selected. `MutualProblematic` denotes that if any one of the set of features is selected then all other features *should preferably not* be selected.

3. Configuration Research Tool

Based on the meta-model presented in Section 2 we developed VISIT-FC, a Visual and Interactive Tool for Feature Configuration. Well known visualisation and interactive techniques were employed to attempt to fulfil MacKinlay's [7] expressiveness criteria. To this end, the VISIT-FC tool strives to display all the information that is required for a particular stakeholder without showing that which can lead to incorrect interpretations through mis-associations. In addition, VISIT-FC adds interactive functionality allowing clear exploration and manipulation of the data.

We have used VISIT-FC to support the product configuration of a Restraint System Control Unit (RESCU) product line. RESCU comprises features for automotive restraint systems such as airbags, seatbelt tensioners, active headrests, and weight sensing. VISIT-FC employs visualisation and interactive techniques that facilitate the product configuration process. It provides a compact, interactive representation of large feature hierarchies, allows configuration with automatic constraint propagation, and provide hints for configuration problems and open decisions.

In the remainder of this section we discuss specific visualisation techniques that are utilised in the tool. The techniques are explicit representation, horizontal linear tree layout, details on demand, incremental browsing, and focus+context.

3.1. Explicit Representation

VISIT-FC uses Explicit Representation as opposed to Implicit Representation. Explicit Representation refers to drawing methods which display the hierarchy as links between nodes. Implicit drawing methods represent the hierarchy by a special arrangement of nodes, e.g. containment or overlapping. Examples of implicit graph drawing are tree-maps [8], or the information cube [14]. Figure 2a shows a screenshot of main RESCU product line features in VISIT-FC.

3.2. Horizontal Linear Tree Layout

Advanced layouts exist for explicit tree-drawings such as cone-trees or space-trees [10]. However, for the purpose of this prototype, a 2D visualisation was chosen, and therefore a simple non-radial tree layout [9] was adopted. The horizontal orientation is preferable over the vertical orientation although the tool does allow the stakeholder to view the model in vertical tree layout. The non-radial (linear) layout and horizontal orientation combine to provide the optimal use of screen space to allow the display of the kinds of data related to a product line feature model. As an example, displaying the names of features on screen with a radial or vertical tree layout would result either in large amounts of overlapping or a zoomed out view (to avoid overlapping) both of which would obscurely render the visualisation.

The combination of an Explicit Representation and a Horizontal Linear Tree provides the opportunity to encode a significant amount of information on screen utilising the restricted space in an efficient manner. VISIT-FC uses an explicit horizontal linear tree layout where the nodes represent features and the edges represent the relationships between those features. Straight edges indicate parent-child relationship and curved edges represent dependency relationships. Figure 2b shows a portion of the RESCU feature model.

Colour coding of the features adds another layer of information to this basic node link tree structure. The colours indicate the configuration status of the selected features and their sub-features. A `FeatureGroup` is colour-encoded *mandatory but not configured* if its sub-features are not resolved. There are four levels of colour encoding, one for each of the feature states, which are *selected (green), eliminated (grey), optional (amber)* and *mandatory but not configured (red)*. These colour codes allow a quick overview of the feature model and its state, for instance to see if a valid product configuration exists. Further information is encoded by use of graphical symbols (tick or cross). A tick indicates selection, a cross indicates elimination.

Another layer of information is encoded through the use of additional colour coding. If the box is shaded, then the feature has been pre-configured or eliminated at an earlier stage of configuration and is no longer changeable. If the box is not shaded but the icon is not coloured, then the feature was selected or eliminated based on a dependency. Information encoded at this low level of visual representation is processed pre-attentively [10] by the human graphical system. Therefore once the colour encoding becomes familiar, a stakeholder would be able to interpret large representations rapidly.

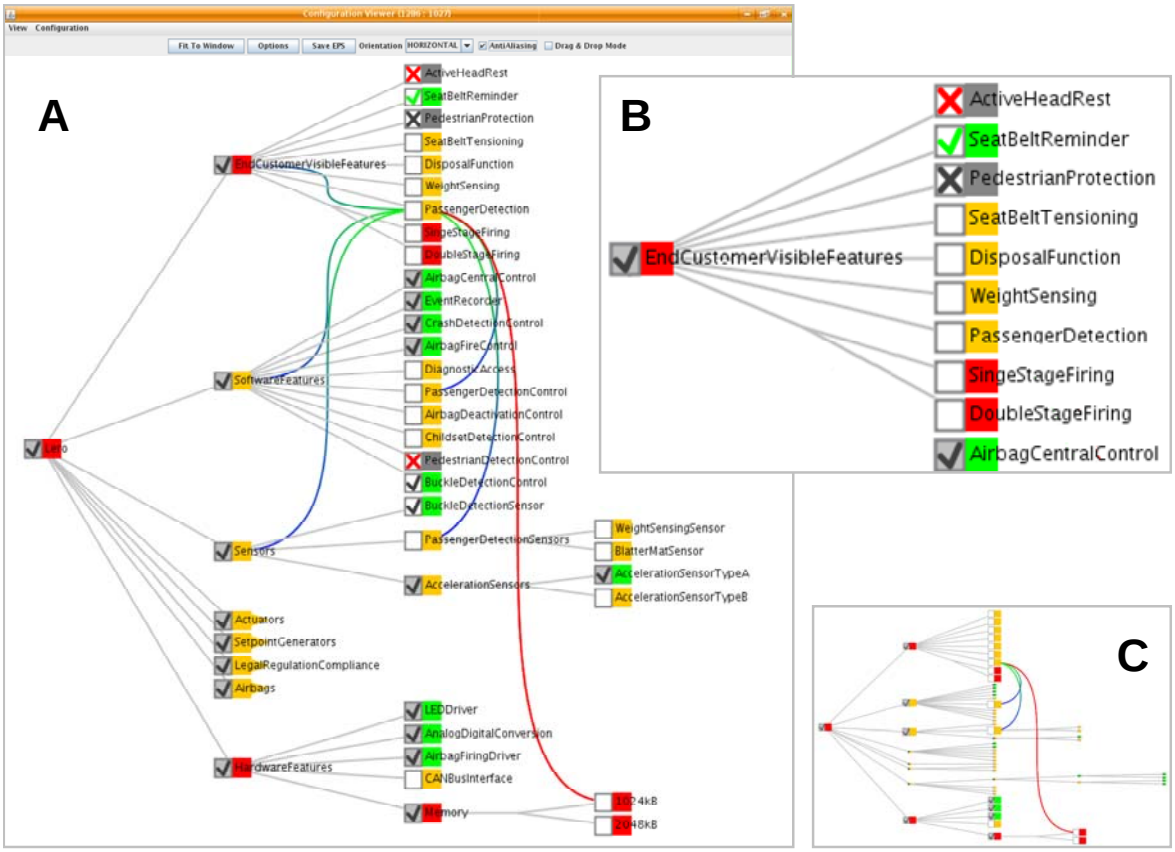

Figure 2. VISIT-FC Configuration Viewer Showing Features of the RESCU Product Line

3.3. Details on Demand

Details on Demand refer to the facility whereby the stakeholder can choose to display additional detailed information at a point where this data would be useful. Information such as cardinalities can be displayed through the use of a "mouse-over" and feature names can be displayed or removed through viewing configuration options.

VISIT-FC also provides the facility to choose a specific feature and show all sub features and dependent features while hiding all other features that are neither sub features nor dependent in any way on the chosen feature. This allows the stakeholder to focus on the relevant data for a particular feature while temporarily removing irrelevant data.

3.4. Incremental Browsing

Incremental browsing is a form of information filtering, where only limited sections of the visualised structure are displayed. The rest is hidden and can be visualised when needed.

In VISIT-FC the feature model visualisation starts with displaying only the high-level features, and the stakeholder can then explore the feature hierarchy by unfolding the sub-features of features in which the stakeholder is interested in. The stakeholder is thus able to perceive the feature structure step by step, and is not overwhelmed by the complete model.

3.5. Focus+Context

Focus+Context refers to the ability to focus on a particular aspect or portion of the visualisation while not losing the context in which that aspect or portion resides [11]. The advantage of Focus+Context is that the stakeholder does not get lost when zooming into a large structure, or exploring the details of certain features. They are always able to see where they came from, and are not required to keep this in memory. This can be useful, e.g., for the visualisation of search results or to see dependent feature nodes in distant parts of a large model.

Pan, Zoom and Degree of Interest in combination are powerful techniques that allow the stakeholder to move around the visualisation, zoom and highlight a

particular area of interest. VISIT-FC provides these facilities and also allows selective zooming of a specific chosen portion of the feature tree focusing on the area of interest and allowing the non-relevant area to remain in view but to a lesser degree. Figure 2c shows a simplified version to illustrate the split zooming facility. It shows certain user selected features that have been "zoomed out" because they are of lesser interest while keeping them in view which maintains the overall context. Different sets of feature nodes can be "zoomed in" or "zoomed out" to varying degrees to allow an optimum view for the task at hand.

4. Feature Configuration Example

To illustrate the use of VISIT-FC and its benefits, this section describes an example that a stakeholder would undertake to configure the diagnosis interface of the RESCU product line.

In this scenario, the stakeholder is interested in configuring "Diagnostic Access" (see the corresponding green node in Figure 3). By clicking on the Diagnostic Access node, the stakeholder can select this feature for the product being derived. On selection, the application automatically configures two other features in the product line by selecting the feature "CAN Bus Interface" (a sub-feature of "Hardware Features") and eliminating the "1024KB Memory" variant. These dependent features are highlighted through increased node size notifying the stakeholder of the automatic actions. If a dependent node is not currently displayed at the point of automatic selection / elimination, then it is made visible at that time. The stakeholder can then distinctly display the dependencies using curved colour coded links. By use of split zooming and panning, the stakeholder modifies the display for even further clarity. If desired the stakeholder can display all dependent features providing a useful view of connected parts of the product being derived. Moreover, he or she can switch the view to the dependency context mode temporarily removing all data from the screen except that which is directly connected to the feature being configured.

5. Related Work

FeaturePlugin [12], pure::variants [13], COVAMOF [14] and Gears [15] are examples of other feature modelling tools that employ a visual representations to aid product configuration and variability management.

FeaturePlugin is an Eclipse plug-in that supports feature modelling. It employs nested lists and a tree layout to support product configuration. Some of the drawbacks of FeaturePlugin are that the lists can be difficult to navigate as the focus+context display implementation is not very effective. It is also difficult to comprehend the dependencies as constraints are shown as unsorted lists.

pure::variants is a product configuration package developed by pure-systems GmbH. It supports various views which provide different configuration approaches for stakeholder tasks but does not support cardinality. Using the built in automatic layout, can adversely affect the tree layout which can be confusing. Large industrial product lines could easily lead to information overload.

COVAMOF and Gears are further tool suits that support feature and product configuration. Even though significant functionality exists, they lack in visual support which makes the understanding of the overall configuration state very difficult.

6. Future Work

The development of the VISIT-FC prototype is based on the utilisation of well understood but non-complex visualisation and interaction techniques. It has shown an avenue down which the challenges faced by stakeholders during product configuration can be addressed. Even simple information encoding through colour schemes suggests an increase in the speed at which product configurations can be interpreted. More in depth research into visualisation techniques and their applicability to and usability for, variability management tasks is planned.

Development of the tool to implement further functionality provided by the meta-model is also planned, such as implementation of the `FeatureReference` entity, cloning of features and linking of the asset base, feature model and realisation artefacts. This would provide an end-to-end visual support for an interactive product derivation tool.

7. Conclusions

We have presented a feature configuration meta-model and introduced a prototype tool that utilises the meta-model and employs a variety of visualisation and interaction techniques. We suggest that targeted use of these techniques in combination with a suitable meta-model can provide significant aid to product configuration stakeholders.

To the authors' opinion, further research into the applicability of various visualisation and interaction techniques could significantly increase the efficiency of configuration tasks during product line engineering.. Furthermore, a configurable visualisation toolkit could replace the dependence on a small number of experts and allow software product line engineers execute their tasks with much greater autonomy.

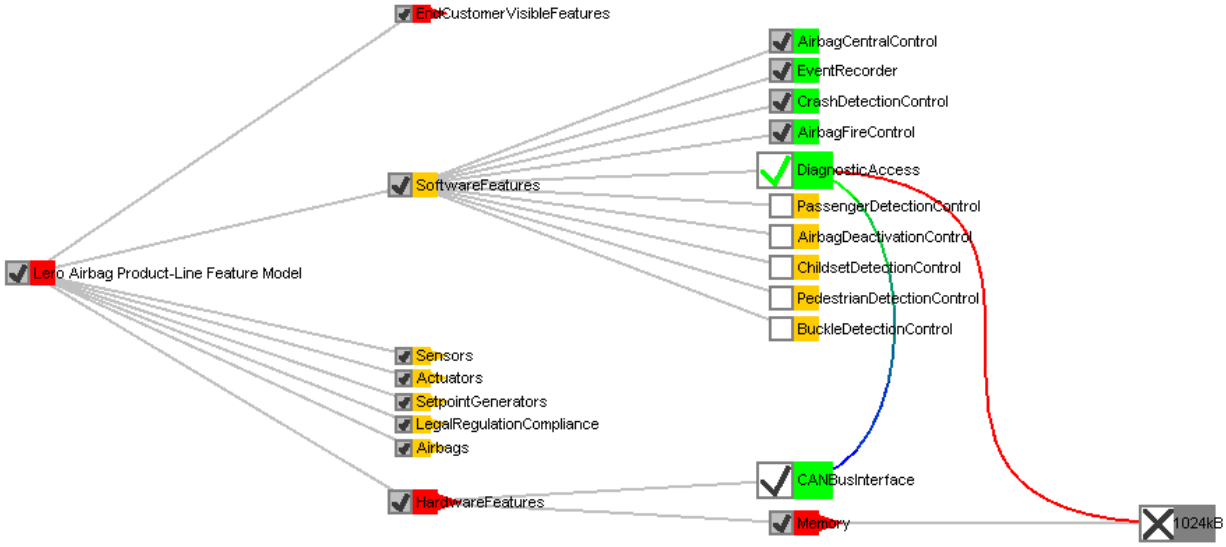

Figure 3. Feature Configuration Example

8. Acknowledgements

This work is partially supported by Science Foundation Ireland under grant number 03/CE2/I303-1.

9. References

[1] K. Pohl, G. Böckle, and F. v. d. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*, 1st ed. New York, NY: Springer, 2005.

[2] M. Sinnema and S. Deelstra, "Classifying variability modeling techniques," *Information and Software Technology*, vol. 49, pp. 717-739, 2007.

[3] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber, "Introducing PLA at Bosch Gasoline Systems: Experiences and Practices," in *SPLC 2004*, Boston, MA, USA, 2004, pp. 34-50.

[4] S. Deelstra, M. Sinnema, and J. Bosch, "Product Derivation in Software Product Families: A Case Study," *Journal of Systems and Software*, vol. 74, pp. 173-194, 2005.

[5] C. Cawley, D. Nestor, A. Preußner, G. Botterweck, and S. Thiel, "Interactive Visualisation to Support Product Configuration in Software Product Lines," *Proceedings of 2nd International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS 2008)*, Essen, Germany, January 16-18, 2008.

[6] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged Configuration Using Feature Models," in *Proceedings of SPLC 2004*, 2004, pp. 266-283.

[7] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, pp. 110-141, 1986.

[8] B. Johnson and B. Shneiderman, "Tree-maps: a space-filling approach to the visualization of hierarchical information structures," in *Proceedings of the 2nd Conference on Visualization*, 1991, pp. 284–291.

[9] E. M. Reingold and J. S. Tilford, "Tidier Drawings of Trees," *IEEE Transactions on Software Engineering*, vol. 7, pp. 223-228, 1981.

[10] C. Ware, *Information Visualisation: Perception for Design*, 2nd ed.: Morgan Kaufmann, 2004.

[11] S. K. Card and D. Nation, "Degree-of-interest trees: A component of an attention-reactive user interface," *Advanced Visual Interface*, pp. 231-245, 2002.

[12] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature Modeling plug-in for Eclipse," in *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, Vancouver, BC, Canada, 2004, pp. 67--72.

[13] pure-systems GmbH, "Variant Management with pure::variants," http://www.pure-systems.com, Technical White Paper, 2003-2004.

[14] M. Sinnema, O. d. Graaf, and J. Bosch, "Tool Support for COVAMOF," in *SPLC 2004, Workshop on Software Variability Management for Product Derivation* Boston, MA, USA, 2004.

[15] Biglever Software, Gears,: http://www.biglever.com.