

# Interactive Visualisation to Support Product Configuration in Software Product Lines

Ciarán Cawley<sup>1</sup>, Daren Nestor<sup>1</sup>, André Preußner<sup>2</sup>, Goetz Botterweck<sup>1</sup>, Steffen Thiel<sup>1</sup>

<sup>1</sup>Lero, University of Limerick  
Limerick, Ireland  
{ ciaran.cawley | daren.nestor |  
goetz.botterweck | steffen.thiel }@lero.ie

<sup>2</sup>BTU Cottbus  
Institute of Computer Science  
Cottbus, Germany  
apreussn@informatik.tu-cottbus.de

## Abstract

*Software Product Line engineering allows companies to realise significant improvements in time-to-market, cost, productivity, and system quality. One major difficulty with software product lines is that within industry there may exist thousands of variation points in a single product line. This scale of variability can become extremely complex to manage resulting in a product configuration process that bears significant costs. This paper presents a feature configuration meta-model and introduces a prototype tool that employs visualisation and interaction techniques to provide feature configuration functionality.*

## 1. Introduction

Software Product Line (SPL) engineering is a paradigm to develop software applications using platforms and mass customisation. This is achieved through the identification and control of the applications' commonality and variation. Developing using a product line 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 management of such a product line's variability is fundamentally 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 producing a new product from existing product line assets [2].

Within industry, product lines exist with thousands of variation points and configuration parameters that need to be managed in order to customise a 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].

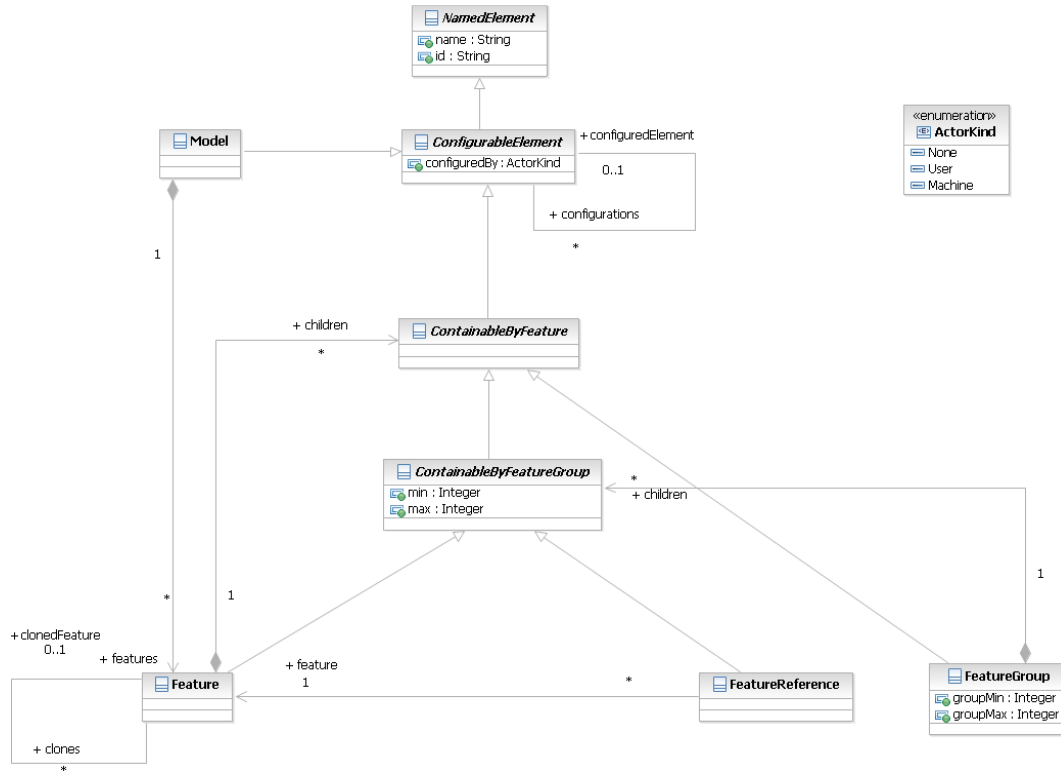
One technique that can be applied beneficially in this context is visualisation. Visualisation takes abstract data and transforms it into a format that is useful for presentation to humans. In doing this, human cognition is enhanced and understanding is afforded. In the area of software product line 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 software 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 prototype tool (VISIT-FC) which is based on the meta-model and discuss the tool's architecture design; in Section 4 we explain some of the visualisation techniques implemented in VISIT-FC and how they help to address the challenges of high variability in large feature models. Section 5 provides an illustrating example of a feature configuration using VISIT-FC. Section 6 discusses related work in visual feature configuration and Section 7 outlines future work. Finally, Section 8 concludes the paper.

## 2. Feature Modelling

A key aspect of the Software Product Line engineering approach is the modelling of the variability of the supported product line features. Such a feature model can support the derivation of a product allowing the



**Figure 1. Basic structure of the feature model**

inclusion and exclusion of various features and variants so that a valid feature configuration is produced. A feature model can also act as a guide for product configuration and can be used to validate a particular configuration for conformance.

We summarise a meta-model which can be used to describe feature models and which forms the basis of the prototype tool presented in sections 3, 4 and 5.

There are a number of suggested feature modelling languages in existence [6-8] but we have chosen to extend and modify Czarnecki's meta-model [8] (see Figure 1). The following are the reasons behind our extensions and modifications.

- To reduce complexity, explicit reference to SolitaryFeatures, GroupFeatures and RootFeatures was removed and there is no separation between elements that can be contained by a Feature and a FeatureGroup. These aspects of the meta-model presented in [8] are not required for our purposes.
- We needed enhanced support for the product configuration process and so increased the options available for relating features with architecture and for supporting inter feature dependencies.

- Support for the cloning of features within a feature group was added.

The following sub-sections describe the main characteristics of our 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 subclassing 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

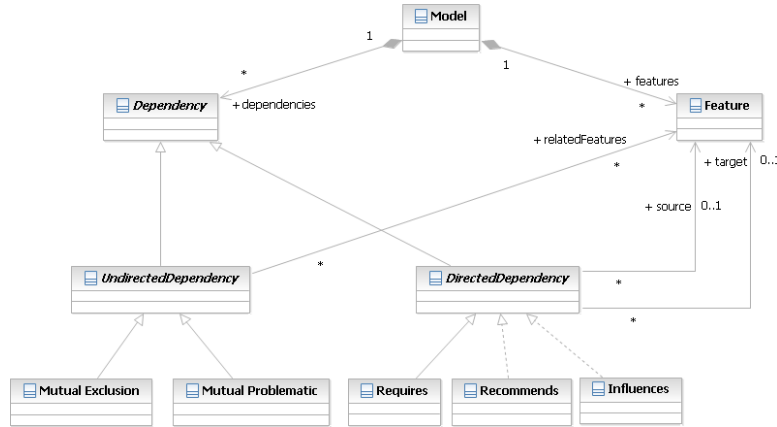


Figure 2. Dependencies among features

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 attributes set to [0,1].

## 2.3. Dependencies

The meta-model supports two types of feature relationships (Figure 2), an UndirectedDependency and a DirectedDependency. Two concrete implementations of a DirectedDependency 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.

Two concrete implementations of an UndirectedDependency are 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. Tool Prototype

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 [10] expressiveness criteria. This criteria states that a set of facts is *expressible* if all the facts in the set, and only the facts in the set, are expressed. 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.

Visualisation has been described as an “adjustable mapping from data to visual form” [11]. The term “map shock” describes a phenomenon whereby a perceiver has an audible reaction to a visual form that displays an overly complex diagram. Visualisations (and VISIT-FC) aim to relate as much relevant information as possible while avoiding such a reaction from a perceiver. In addition, VISIT-FC adds interactive functionality allowing clear exploration and manipulation of the data.

An instance of the meta-model presented in Section 2 has been created and is used to illustrate the visualisation and interactive techniques employed by VISIT-FC to support the product configuration functionality. The feature model instance introduced represents the Restraint System Control Unit (RESCU) product line. This product line contains features of electronic control units (ECUs) for automotive restraint systems such as airbags and seatbelt tensioners.

The following two subsections give a breakdown of the design and architecture of the tool to show the underlying model and how it supports our interactive visualisation approach.

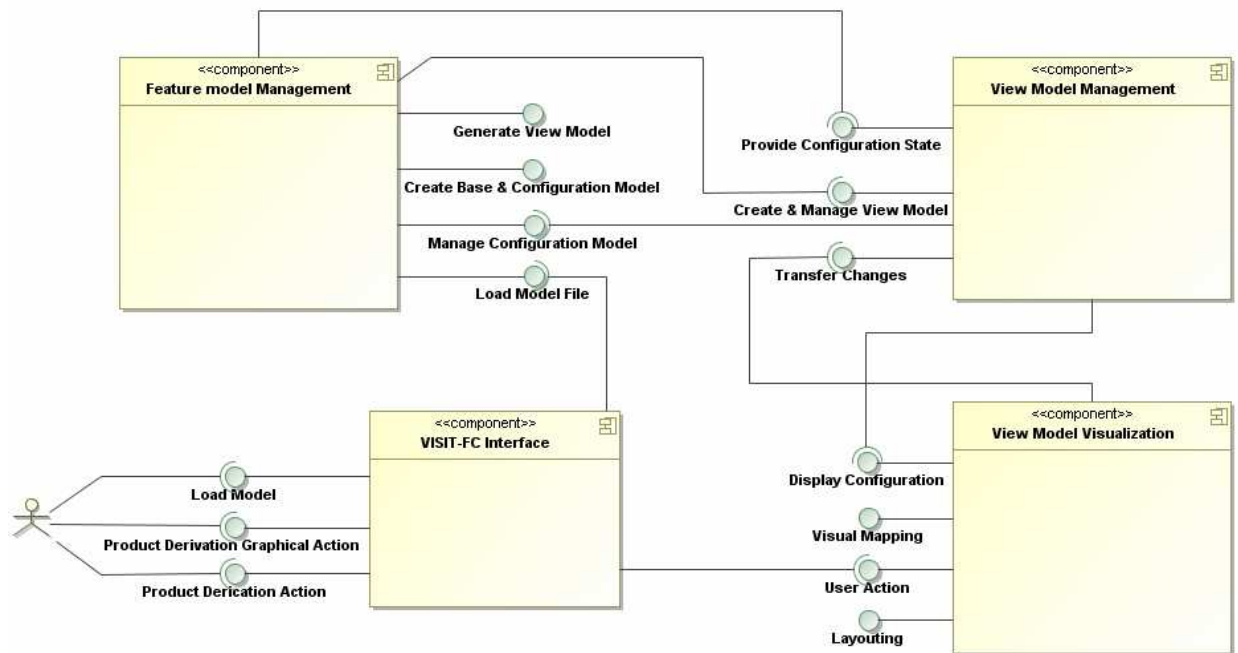


Figure 4. VISIT-FC component diagram

### 3.1. Design Concept

The primary tasks of the VISIT-FC tool are the visualisation of Software Product Line information and feature configuration. The underlying concept of the design is the separation of the various concerns. Feature configuration is divided between the base *Feature Model* and the *View Model* that acts as a broker between the *Feature Model* and a third *Visualisation* component (see Figure 3).

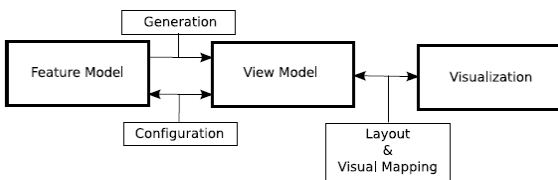


Figure 3. VISIT-FC design overview

As a stakeholder interacts with the *Visualisation*, the *View Model* transfers the information between it and the *Feature Model* and vice versa.

This design affords a number of advantages. As long as a transformation of the *Feature Model* into the *View Model* can be delivered, the *Visualisation* can operate independently of the implementation details of the *Feature Model*. This protects the *Visualisation* against changes to the *Feature Model* and also can allow different feature meta-models to be used. The *View Model*

can hide the complexity of the *Feature Model* and provide a simplified view of it.

At a stakeholder's request, the *Feature Model* component loads an XML file containing the product line feature model. A copy of the model (the configuration model) is generated and each feature is linked between the base model and the copy. The copy is used to record and represent the new state of the feature model at its current stage. The *View Model* brokers the configuration data flow between the *Feature Model* and the *Visualisation*. The *Visualisation* is controlled by layout and mapping mechanisms that use defined information for visualising the contents of the *View Model* and providing interactive functionality to the stakeholder.

### 3.2. Software Architecture

Figure 4 shows the component diagram for the VISIT-FC tool. The tool is comprised of four components.

Firstly, the Feature Model Management component facilitates the loading of the model source XML file, the creation of a *base* and *configuration* model, the generation of the *View Model* and provides the functionality that manages the changes to the model that occur during the product configuration process.

Secondly, the View Model Management component allows the creation of a new *View Model*. It maintains the *View Model* reflecting the product configurations being undertaken, transfers information concerning the

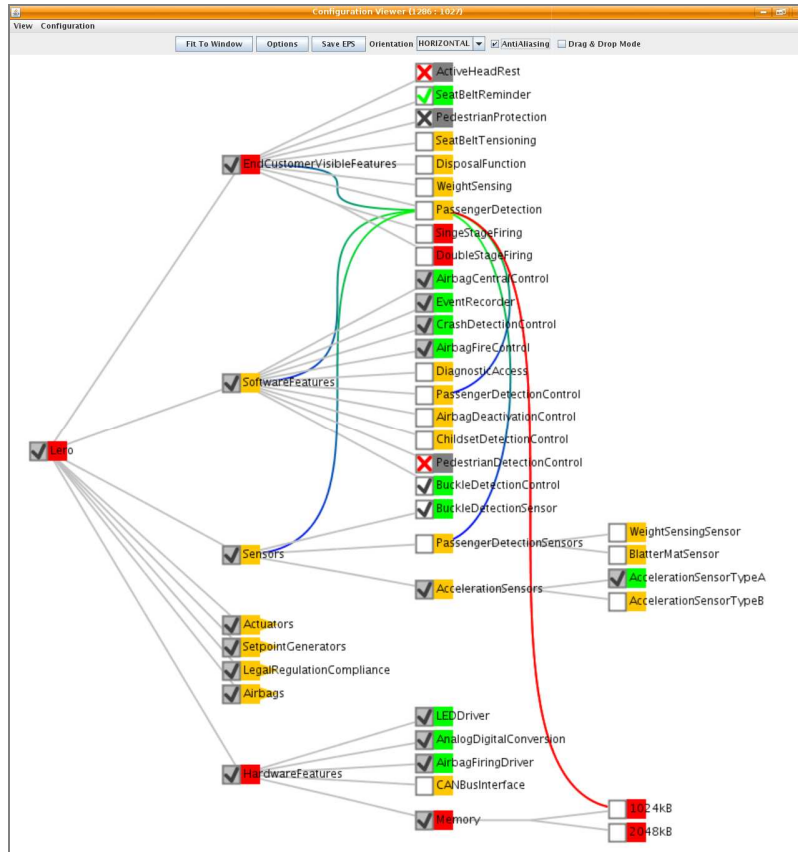


Figure 5. VISIT-FC configuration viewer showing features of the RESCU product line

configuration model to the View Model Visualisation component and synchronises the modifications performed by the stakeholder with the Feature Model Management component.

Thirdly, the View Model Visualisation component stores and maintains all the pertinent information related to the graphical representation. It manages the layout algorithms and facilitates graphical manipulations by the stakeholder.

Finally the VISIT-FC User Interface component allows the actual screen events to take place providing functionality such as feature selection, pan and zoom and node manipulation.

To load a model file, functionality within the Feature Model Management component is invoked. The Feature Model component creates a base and configurable model and also generates the *View Model* through a call to the View Model Management component. Finally, a call is made to the View Model Visualisation component resulting in the rendering of the *View Model* visually.

When product configuration actions take place, functionality within the View Model Visualisation

component is invoked that produces a transfer of information between the View Model Visualisation component and the Feature Model Management component through the View Model Management component. When the Feature Model Component receives this information it applies the modification on the configuration model and invokes the View Model Visualisation component via the View Model Management component to display the modifications.

#### 4. Visualisation & Interactive Techniques

The VISIT-FC tool (see Figure 5) aims to employ visualisation and interactive techniques that facilitate the following goals: provide a compact, interactive representation of large feature hierarchies; provide facilities to restrict the view to feature model parts of interest; allow feature configuration with automatic constraint propagation and provide hints for configuration problems and open decisions, see also [5]. This section describes the techniques that are utilised and the specific functionality that is then possible allowing the fulfilment of the specific goals.

#### 4.1. Explicit Representation

The VISIT-FC tool uses Explicit Representation as opposed to Implicit Representation. Explicit Representation refers to drawing methods which display the hierarchy as links between nodes, e.g. [12]. 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 [13], or the information cube [14]. Figure 5 shows a screenshot of the main visualisation of the RESCU product line feature model in VISIT-FC.

#### 4.2. Horizontal Linear Tree Layout

Advanced layouts exist for explicit tree-drawings such as cone-trees [15] or space-trees [12]. However, for the purpose of this prototype, a 2D visualisation was chosen, and therefore a simple non-radial tree layout [16] 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 gives us 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.

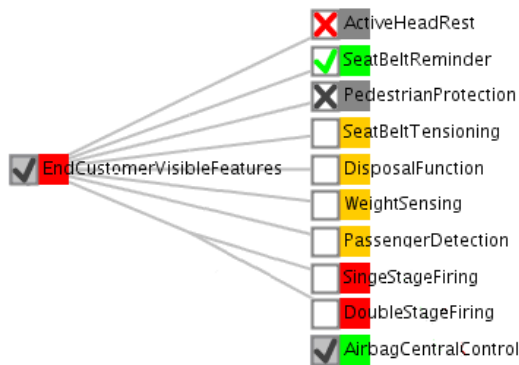


Figure 6. Information encoding in VISIT-FC

Figure 6 shows a portion of the RESCU feature model illustrating the information that is encoded in quite a small screen area. 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, that is, 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. To this, 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 [11] [17] by the human graphical system. Therefore once the colour encoding becomes familiar, a stakeholder would be able to interpret large representations rapidly.

#### 4.3. 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” (see Figure 7) and feature names can be displayed or removed through viewing configuration options.

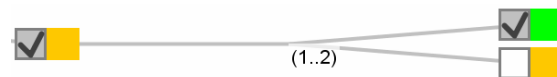


Figure 7. Relationship showing cardinalities

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 (see Figure 8).

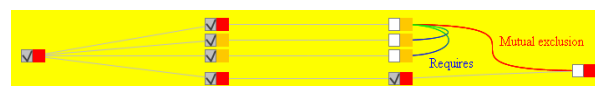


Figure 8. Contextual view

This allows the stakeholder to focus on the relevant data for a particular feature while temporarily removing irrelevant data. This function is easily accessible and removable by use of a keyboard shortcut and mouse movement.

#### 4.4. 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.

Figure 9 is a simple illustration where only one feature has been unfolded. The triangular extension of the colour coded feature indicates further unfolding is possible.

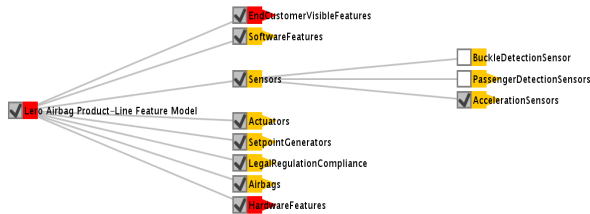


Figure 9. Support for incremental browsing

#### 4.5. 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 e.g. [18]. 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 visualization of search results or to see dependent feature nodes in distant parts of a large feature model within the context of the whole feature structure.

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 10 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.

### 5. Feature Configuration Example

To illustrate the use of VISIT-FC and its benefits, this section describes the steps that a stakeholder would undertake to configure a specific aspect of interest within the RESCU product line. In this instance a stakeholder wishes to explore the “Software Features” of a product configuration that is in progress and add configuration for Diagnostic Access.

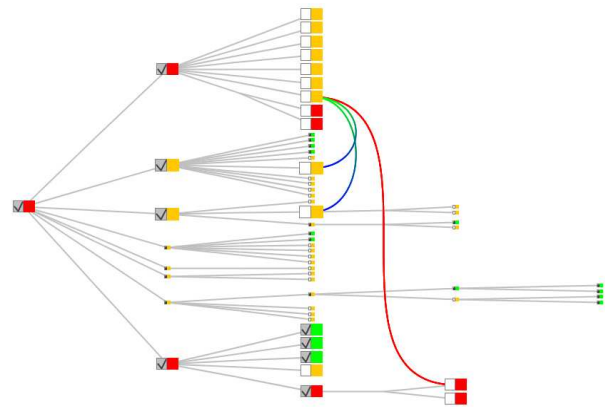


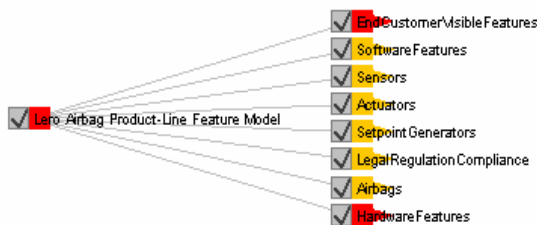
Figure 10. Focus+Context, Degree of Interest and Details on Demand

On start up VISIT-FC shows the root node of the feature model, which can be expanded to show the next level of tree nodes (see Figure 11). Immediately the stakeholder can see that the model has been through a previous configuration stage, that there are eight feature sets, two of those are mandatory and that none of the feature sets have been fully configured as yet. By expanding the “Software Features” node, the stakeholder can explore this section of the product line while keeping all other sections out of the way but still in context. The stakeholder can see that four of the ten sub-features have been configured at a previous stage and that six remain to be resolved.

In this scenario, the stakeholder is interested in configuring “Diagnostic Access” and can see it has not been previously configured (see the corresponding green node in Figure 12). 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 of the feature, 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 by the stakeholder, he can display all other features that are connected in a dependent fashion 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 (Figure 8) temporarily removing all data from the screen except that which is directly connected to the feature being configured.



**Figure 11. Initial Feature Model View**

## 6. Related Work

FeaturePlugin [19], pure::variants [20], COVAMOF [21] and Gears [22] are examples of other feature modelling tools that employ a visual component to aid product configuration and variability management.

FeaturePlugin is an Eclipse IDE plug-in that supports feature modelling. It uses the Eclipse Modelling Framework (EMF) to generate the editors that facilitate the modelling aspect and provides a rich set of modelling functionality. Nested lists and a tree layout are employed as techniques to support product configuration using the FODA style. 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 was developed by pure-systems GmbH and is a software package that provides similar functionality to FeaturePlugin. It supports various views which provide different approaches for different stakeholder tasks but does not support cardinality. Using the built in automatic layout, can adversely affect the tree layout which as it is can be confusing. Large industrial product lines could easily lead to information overload.

A suite of tools exist that provides the implementation of the requirements of the ConIPF Variability Modelling Framework, COVAMOF. Even though significant functionality exists, understanding of the overall state of the configuration can be difficult due to the separate and disconnected window views.

## 7. 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.

The possibility of providing this prototype tool as an Eclipse plug-in will also be explored.

## 8. Conclusions

We have presented a feature configuration meta-model and introduced a prototype tool that utilises that 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.

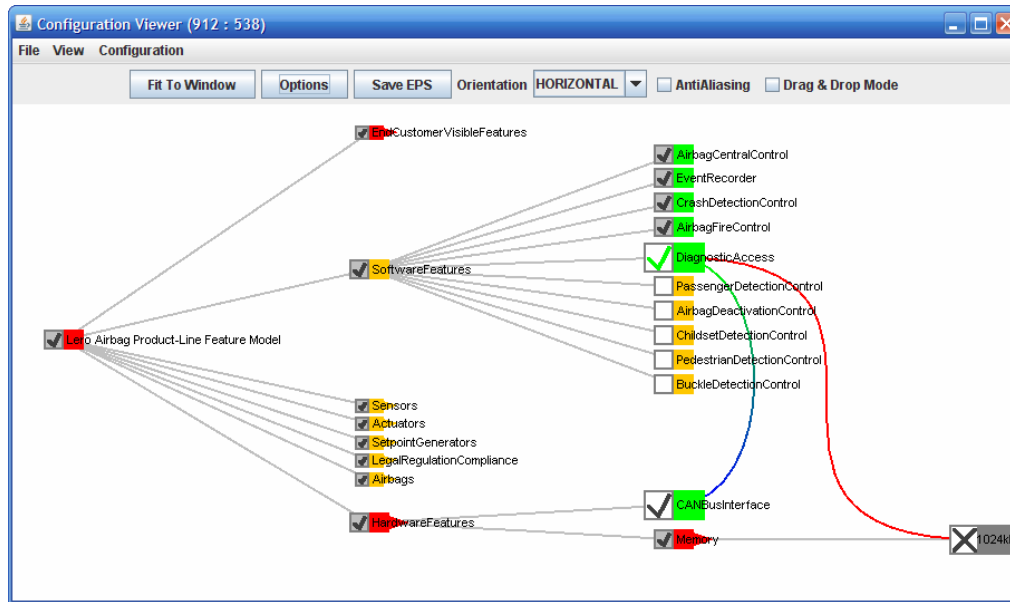


Figure 12. Staged feature configuration example

In the authors' opinion, further research into the applicability of various visualisation and interaction techniques to variability management could significantly lessen the challenges faced by stakeholders and greatly increase the efficiency of a number of tasks that require fulfilment when undertaking 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.

## 9. Acknowledgements

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

## 10. 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] D. Nestor, L. O'Malley, A. Quigley, E. Sikora, and S. Thiel, "Visualisation of Variability in Software Product Line Engineering," in *VaMoS-2007*, Limerick, Ireland, 2007.
- [6] K. Kang, S. Kim, J. Lee, K. Kim, S. E., and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, vol. 5, pp. 143-168, 1998.
- [7] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," SEI, Carnegie Mellon University CMU/SEI-90-TR-21, November 1990.
- [8] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged Configuration Using Feature Models," in *Proceedings of SPLC 2004*, 2004, pp. 266-283.
- [9] T. Bednasch, "Konzept und Implementierung eines konfigurierbaren Metamodells für die Merkmalmodellierung," in *Fachbereich Informatik* Zweibrücken, Germany: Fachhochschule Kaiserslautern, 2002.
- [10] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, pp. 110-141, 1986.
- [11] S. K. Card, J. D. MacKinlay, and B. Shneiderman, *Readings in Information Visualization - Us-*

- ing Vision to Think*, 1st ed. San Francisco: Morgan Kaufmann Publishers, 1999.
- [12] C. Plaisant, J. Grosjean, and B. B. Bederson, "SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation," in *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2002)*, 2002.
  - [13] 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.
  - [14] J. Rekimoto and M. Green, "The information cube: Using transparency in 3d information visualization," in *Workshop on Information Technologies & Systems*, 1993, pp. 125-132.
  - [15] G. G. Robertson, J. D. Mackinlay, and S. K. Card, "Cone trees: Animated 3D visualizations of hierarchical information," *Proceedings of CHI 9*, pp. 189-194, 1991.
  - [16] E. M. Reingold and J. S. Tilford, "Tidier Drawings of Trees," *IEEE Transactions on Software Engineering*, vol. 7, pp. 223-228, 1981.
  - [17] C. Ware, *Information Visualisation: Perception for Design*, 2nd ed.: Morgan Kaufmann, 2004.
  - [18] 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.
  - [19] 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.
  - [20] pure-systems GmbH, "Variant Management with pure::variants," <http://www.pure-systems.com>, Technical White Paper, 2003-2004.
  - [21] 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.
  - [22] Biglever Software, Gears,; <http://www.biglever.com>.