Towards Multi-view Feature-Based Configuration

Arnaud Hubaux[1], Patrick Heymans[1], Pierre-Yves Schobbens[1], and Dirk Deridder[2]

[1] PReCISE Research Centre, Faculty of Computer Science, University of Namur
Namur, Belgium
{ahu,phe,pys}@info.fundp.ac.be
[2] Software Languages Lab, Vrije Universiteit Brussel,
Brussels, Belgium
Dirk.Deridder@vub.ac.be

Abstract. [Context & motivation] Variability models, feature diagrams ahead, have become commonplace in software product line engineering as a means to document variability early in the lifecycle. Over the years though, their application span has been extended to aid stakeholders in the configuration of software products. **[Question/problem]** However, current feature-based configuration techniques hardly support the tailoring of configuration views to the profiles of heterogeneous stakeholders. **[Principal ideas/results]** In this paper, we introduce a lightweight mechanism to leverage multidimensional separation of concerns in feature-based configuration. **[Contribution]** We propose a technique to specify concerns in feature diagrams and to build automatically concern-specific configuration views, which come with three alternative visualisations.

1 Introduction

An increasing number of software developments adopt the paradigm of software product line engineering (SPLE) [1]. The goal of SPLE is to rationalise the development of families of similar software products. A key idea is to institutionalise reuse throughout the development process to accomplish economies of scale.

SPLE is a very active research area at the crossroads between many software development related disciplines, including requirements engineering (RE). An important research topic in SPLE and RE is feature diagrams (FDs) [2,3]. FDs are a simple graphical formalism whose main purpose is to document variability in terms of *features*, i.e. high-level descriptions of the capabilities of the reusable artefacts.

FDs have been given a formal semantics [3], which opened the way for safe and efficient automation of various, otherwise error-prone and tedious, tasks including consistency checking [4,5], decision propagation [4] and process control [6,7,8]. A repertoire of such automations can be found in [9]. The kind of automation that we focus on in this paper is feature-based configuration (FBC). FBC is an interactive process during which one or more stakeholders select and discard features to build specific products. FBC is one of the principal means to elicit product requirements in SPLE. In practice, there can be thousands of features whose legal combinations are governed by many and often complex rules [4]. It is thus of crucial importance to be able to simplify and automate the decision-making process as much as possible.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 106–112, 2010.

Two challenges that FBC techniques fail to address in a satisfactory way are (1) *tailoring the configuration environment* according to the stakeholder's profile (knowledge, role, preferences...) and (2) *managing the complexity* resulting from the size of the FD.

In this paper, we outline a solution strategy to address those two challenges. We do so by extending FDs with multiple views that can be used to automatically build FD visualisations. A view is a streamlined representation of a FD that has been tailored for a specific stakeholder, task, or, to generalize, a combination of such elements—which we call a *concern*. Views facilitate configuration in that they only focus on those parts of the FD that are relevant for a given concern. Using multiple views is thus a way to achieve *separation of concerns* (SoC) in FDs. SoC helps making FD-related tasks less complex by letting stakeholders concentrate on the parts that are relevant to them while hiding the others. Further tailoring of the visualisations is suggested through the selection of three alternative visualisations: (1) "greyed out", (2) "pruned" and (3) "collapsed".

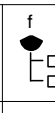
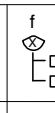
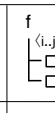
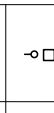
In the rest of this paper, we elaborate on these ideas. Section 2 introduces FDs. A motivating example is given in Section 3. Section 4 presents our basic strategy for constructing views.

2 Feature Diagram

Schobbens *et al.* [3] defined a generic formal semantics for a wide range of FD dialects. We only recall the basic concepts. In essence, a FD d is a hierarchy of features (typically a tree) topped by a root feature. Each feature has a cardinality $\langle i..j \rangle$ attached to it, where i (resp. j) is the minimum (resp. maximum) number of children (i.e. features at the level below) required in a product (aka configuration). For convenience, common cardinalities are denoted by Boolean operators, as shown in Table 1. Additional constraints that crosscut the tree can also be added and are defined, without loss of generality, as a conjunction of Boolean formulae. The semantics of a FD is the set of products. The full syntax and semantics as well as benefits, limitations and applications of FDs are extensively discussed elsewhere [3,9].

FBC tools use FDs to pilot the configuration of customisable products. These tools usually render FDs in an *explorer-view* style [10,4], as in the upper part of Table 1. The tick boxes in front of features are used to capture decisions, i.e. whether the features are selected or not. We now illustrate this more concretely with a motivating example.

Table 1. FD decomposition operators

Concrete syntax					
Boolean operator	and: $\wedge$	or: $\vee$	xor: $\oplus$	non standard	optional
Cardinality	$\langle n..n \rangle$	$\langle 1..n \rangle$	$\langle 1..1 \rangle$	$\langle i..j \rangle$	$\langle 0..1 \rangle$

3 Motivating Example

Spacebel is a Belgian software company developing software for the aerospace industry. We collaborate with Spacebel on the development of a SPL for flight-grade libraries implementing the CSSDS File Delivery Protocol (CFDP) [8]. CFDP is a file transfer protocol designed for space requirements, such as long transmission delays and specific hardware characterised by stringent resource limitations. Spacebel built a SPL of CFDP libraries, where each library can be tailored to the needs of a specific space mission.

The FD of the CFDP SPL counts 80 features, has a maximal depth of four and contains ten additional constraints. A simplified excerpt of this FD appears in the upper part of Figure 1[1]. The principal features provide the capability to send (*Send*) and receive (*Receive*) files. The *Extended* feature allows a device to send and receive packets via other devices. The *Reboot* feature allows the protocol to resume transfers safely after a sudden system reboot. PUS stands for Packet Utilisation Standard, part of the ESA

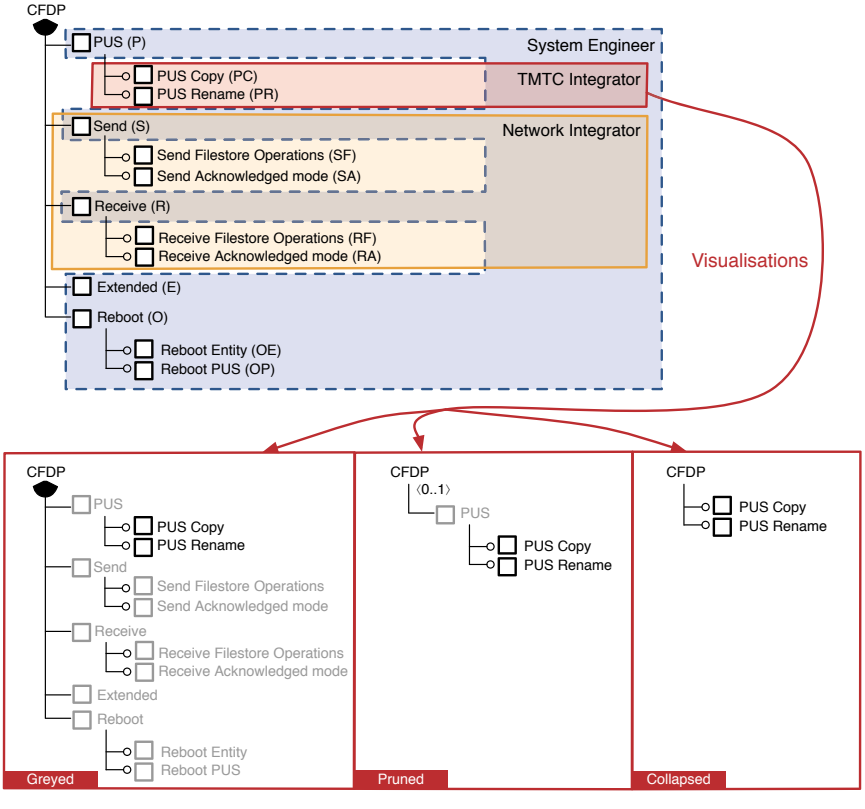


Fig. 1. FD of the CFDP with three alternative visualisations for the view of the TMTC integrator

[1] An online version designed with SPLOT, an open source web-based FBC tool, is available at `http://www.splot-research.org/`.

standard for transport of telemetry and telecommand data (TMTC). The *PUS* feature implements the CFDP related services of this standard.

CFDP typically handles four different stakeholder profiles. *Spacebel* decides which features are mature enough for the mission, while leaving as much variability as possible. The *system engineer* makes initial high-level choices and passes the task of refining these choices on to the *network integrator* and the *TMTC integrator* who handle the technical aspects of the CFDP. The configuration options of interest for each of these profiles are thus different and limited in scope.

A major problem is that access rights to these configuration options are currently informally defined and FDs offer no way to do so. In the absence of clear access specifications, a simplistic policy has been implemented: all profiles have access to all configuration options. A reported consequence is that sometimes the system engineer does not have sufficient knowledge to fully understand low-level options and make decisions. The results were incorrect settings, e.g., inappropriate CPU consumption or excessive use of memory for a given hardware. Similarly, the integrators were not aware of general decisions and could make inconsistent choices wrt. the missions' goals.

The changing context also demands flexible definitions of access policies. For instance, there can be variations in the access rights (e.g., the integrators are granted access to more features) or stakeholder profiles (e.g. a dedicated *File System integrator* might be needed in some projects).

This situation provided the initial motivation for the solution outlined in this paper. However, as we will see, the solution is applicable to a wider variety of problems than the sole definition of configuration access rights. Its ambition is to extend FDs with support for multiple perspectives.

4 Multi-view Feature Diagrams

Solving the problem described in the previous section requires being able to specify which parts of the FD are configurable by whom. This can be achieved easily by augmenting the FD with a set V of views, each of which consists of a set of features. Each view $v_i \in V$ contains some features of the FD. A view can be defined for any concern that requires only partial knowledge of the FD. Also, as a general policy, we consider that the root is part of each view.

View specification. There are essentially two ways of specifying views. The most obvious is to enumerate, for each view, the features that appear in it, or equivalently, to tag each feature of the FD with the names of the views it belongs too. These are *extensional definitions*, which might be very time-consuming and error-prone if the FD is too big and there is no appropriate tool support. A natural alternative is thus to provide a language for *intensional definitions* of views that takes advantage of the FD's tree structure to avoid lengthy enumerations. A simple query language like XPath would be a good candidate to define views and retrieve the corresponding features.

In Figure 1, views have been illustrated by coloured areas. The first area (blue) consists of the high-level features that should be accessible to the system engineer. The last

two areas (red and orange) respectively contain the technical features that should be accessible to the TMTC and network integrators.

View coverage. An important property to be guaranteed by a FBC system is that all configuration questions be eventually answered [7], i.e. that a decision be made for each feature of the FD. A *sufficient condition* is to check that all the features in the FD are in the views of V. The FD of Figure 1 fulfils that condition. But this is not necessary since some decisions can usually be deduced from others.

A *necessary and sufficient condition* can be defined using the notion of propositional defineability [11]. We need to ensure that the decisions on the features that do not appear in any view can be inferred from (are *propositionally defined by*) the decisions made on the features that are part of the view. This can be achieved by translating the FD into an equivalent propositional formula and apply the algorithm described in [11]. Features that do not belong to any view and that do not satisfy the above condition will have to be added to existing views, or new views will have to be created to configure them.

View interactions. Another important property of FBC is that it should always lead to valid configurations [7]. In our case, doing the configuration through multiple views is not a problem *per se*. This is because, although stakeholders only have partial views, the FBC system knows the whole FD and is thus capable of propagating the choices made in one view to the others. However, problems can arise when the selection of a feature in one view depends on the selection of another feature in another view. If overriding of decisions across views is not allowed, then we must introduce some form of conflict resolution mechanisms. This is a complex issue for which various strategies can be elaborated. One is to introduce priorities on views [12]. Another one is to constrain the order in which views are configured [8].

Visualisation. Views are abstract entities. To be effectively used during FBC, they need to be made concrete, i.e. visual. We call a visual representation of a view a *visualisation*. The goal of a visualisation is to strike a balance between (1) showing only features that belong to a concern and (2) including features that are *not* in the the concern but that allow the user to make informed decisions. For instance, the *PUS copy* feature is in the view of the TMTC integrator, but its parent feature *PUS* is not: How will that influence the decision making process? To tackle this problem, we see three visualisation alternatives with different levels of details (see lower part of Figure 1).

The *greyed* visualisation is a mere copy of the original FD in which the features that do not belong to the view are greyed out (e.g. P, S, SF and SA). Greyed out features are only displayed but cannot be manually selected/deselected. In the *pruned* visualisation, features that are not in the view are pruned (e.g. S, SF and SA) unless they appear on a path between a feature in the view and the root, in which case they are greyed out (e.g. P). Pruning can have an impact on cardinalities. As shown in Figure 1, the cardinality of *CFDP* is $\langle 0..1 \rangle$ whereas it is $\langle 1..5 \rangle$ (or-decomposition) in the FD. It has to be recomputed to ensure the consistency of the FD. In the *collapsed* visualisation, all the features that do not belong to the view are pruned. A feature in the view whose parent or ancestors are pruned is connected to the closest ancestor

that is still in the view. If no ancestor is in the view, the feature is directly connected to the root (e.g. PC and PR). Similarly, cardinalities have to be recomputed for consistency reasons.

5 Conclusion

In this paper, we have outlined an approach to specify views on feature diagrams in order to facilitate feature-based configuration, one of the main techniques to define product requirements in software product lines. Three alternative visualisations were proposed, each offering different levels of detail. This work was motivated by an ongoing collaboration with the developers of Spacebel, a Belgian software company developing software for the aerospace industry. A preliminary evaluation with the developers of an open source web-based meeting management system is also in progress [13].

A number of future work can be envisioned. First, a more thorough evaluation should be carried out. Second, we will have to address the problem of conflictual configuration decisions across views. Third, the formalisation needs to be refined and the implementation to be pursued. Currently, we only have standalone algorithms implementing our transformations. The rest of our approach needs to be developed, integrated in a feature modelling and configuration environment, and properly validated.

Acknowledgements

This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy, under the MoVES project.

References

1. Pohl, K., Bockle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (July 2005)
2. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University (November 1990)
3. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature Diagrams: A Survey and A Formal Semantics. In: RE 2006, pp. 139–148 (September 2006)
4. Mendonça, M.: Efficient Reasoning Techniques for Large Scale Feature Models. PhD thesis, University of Waterloo (2009)
5. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
6. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration through specialization and multi-level configuration of feature models. Software Process: Improvement and Practice 10(2), 143–169 (2005)
7. Classen, A., Hubaux, A., Heymans, P.: A formal semantics for multi-level staged configuration. In: VaMoS 2009. University of Duisburg-Essen (January 2009)
8. Hubaux, A., Classen, A., Heymans, P.: Formal modelling of feature configuration workflow. In: SPLC 2009, San Francisco, CA, USA (2009)

9. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. IST (2010) (preprint)
10. pure-systems GmbH: Variant management with pure: variants. Technical White Paper (2006), http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf
11. Lang, J., Marquis, P.: On propositional definability. Artificial Intelligence 172(8-9), 991–1017 (2008)
12. Zhao, H., Zhang, W., Mei, H.: Multi-view based customization of feature models. Journal of Frontiers of Computer Science and Technology 2(3), 260–273 (2008)
13. Hubaux, A., Heymans, P., Schobbens, P.Y.: Supporting mulitple perspectives in feature-based configuration: Foundations. Technical Report P-CS-TR MPFD-000001, PReCISE Research Centre, Univ. of Namur (2010), http://www.fundp.ac.be/pdf/publications/69578.pdf