Concerns and their separation in feature diagram languages: An informal survey

Thein Than Tun Patrick Heymans
PReCISE Research Centre, Faculty of Computer Science
University of Namur, Belgium
{*ttu,phe*}*@info.fundp.ac.be*

***Abstract*—Feature diagrams describe valid configurations of features in a software product line. A major limitation of current feature diagram languages is that they are found not to scale well when applied to realistic software product lines: feature diagrams quickly become too complex to be understood by engineers, and too vague to be analysed by reasoning tool. One well-known design principle for managing complexity is the separation of concerns. However, the nature of important concerns in software product line development, and the extent to which the separation of concerns is addressed by current feature diagram languages are not clear. In this paper, we report on our initial survey of important concerns considered by feature diagram languages and guidelines for addressing those concerns.**

***Keywords*-Software Product Line; Feature Diagrams; Separation of Concerns; Survey;**

I. Introduction

A Feature Diagram (FD) shows possible valid configurations of features within a software product line. In a product line with a realistic number of features, the relationships between features are many and varied. A major limitation of current feature diagram languages is that they are found not to scale well when applied to realistic software product lines: feature diagrams quickly become too complex to be understood by engineers, and too vague to be analysed by reasoning tool [1], [2].

The principle of separation of concerns [3], [4] points to an effective way to manage the size and complexity of FDs [5]. As explained by Dijkstra [3], *separation of concerns* requires a willingness

> ... to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. [...] But nothing is gained—on the contrary!—by tackling these various aspects simultaneously. It is what I sometimes have called "the separation of concerns" ...

Therefore, separation of concerns is about recognising that a system may be decomposed using different criteria [6], and the need to be able to distinguish a decomposition made according to a criterion from another. We believe that such modularisation of feature diagrams can make them scale better.

Two issues are addressed in this paper. First, if correctness and efficiency are some of the main concerns of programs, what are the important concerns of FDs? For instance, FDs may be designed to describe design options, choices of user functionality and legal constraints. There are many other legitimate concerns that should be taken into account in FDs.

Second, having recognised the concerns, what guidelines for addressing those concerns are provided by FD techniques? In this paper, we survey various concerns considered by FD languages, and guidelines for separating concerns.

The rest of the paper is organised as follows. Section II provides an overview of the FD languages surveyed, and a summary of concerns recognised by the surveyed languages and guidelines for addressing those concerns. Discussions and concluding remarks are given in Section III.

II. Languages Surveyed

Although the software product line literature has a rich history, it is somewhat fragmented. This initial survey of FD languages and techniques makes no claim for completeness. We believe that many of the relevant work in the area of SPL research has been covered and are actively seeking to expand and revise our survey. In this paper, we have focused on the following work: FODA [7], FORM [8], Batory *et al.* [9], OVM [10], Staged Configuration [11], Reiser and Weber [1], Metzger *et al.* [12], Hubaux *et at.* [13], and Tun *et at.* [14].

We now give a brief summary of each of the approaches surveyed, roughly in the chronological order of their publication.

A. FODA [7]

Kang *et al.* proposes dividing features into *standard* features, *alternative/optional* features, *specialisation* features, *mutually exclusive* features, and *required* features. They also group features into classes based on their binding time: there are *compile-time*, *load-time* and *run-time* features. Finally, they refer to the four categories of features, discussed in greater detail by Lee *et al.* (shown in Figure 1).

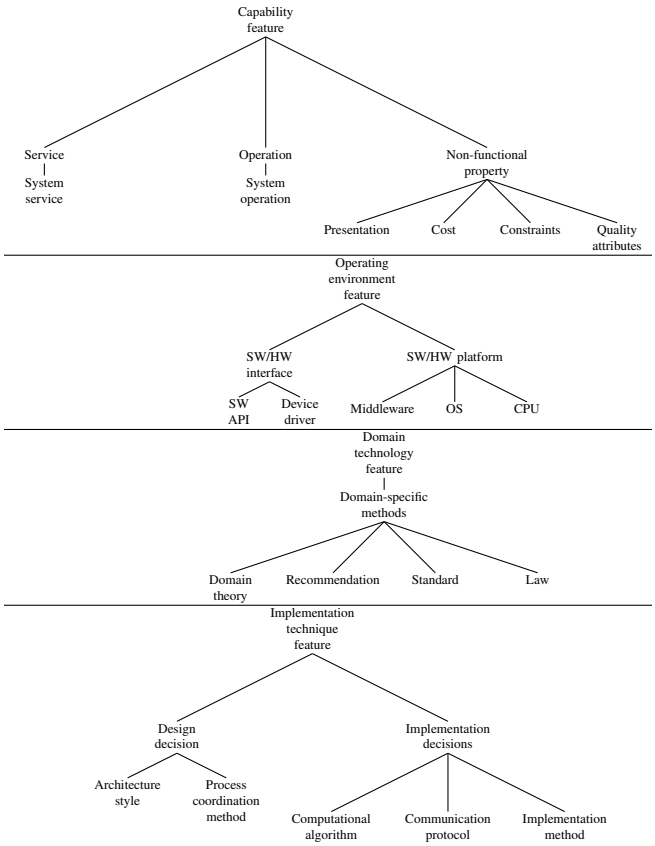

Figure 1. Feature Categories of Lee *et at.* [8]: Capability, Operating Environment, Domain Technology and Implementation Techniques

B. FORM [8], [15]

Lee *et al.* [8], [15] propose several categories of features, organised into a hierarchy as shown in Figure 1. FDs in the capabilities layer address the functionality of the end user; FDs in the operating environments layer address the attributes of the environment in which the application is used; FDs in the domain technologies layer address the application specific non-technical issues, whilst FDs in the implementation technologies layer address technologies that are not specific to a particular domain.

In addition to these categories, Lee *et al.* use *composed-of*, *generalisation/specialisation* and *implemented-by* relationships.

C. Batory et al. [9]

Batory *et al.* [9] proposed an approach for "multi-dimensional separation of concerns" [4], which recognises that features may be partitioned in a number of ways (dimensions) and the results of each partitioning is called *units*[1]. For instance, (object-oriented) classification is regarded as a dimension and classes of a software are units.

[1]Although this approach is focused on features, rather than FDs, the way concerns are separated is of interest to this survey.

They propose using the "origami matrix" for describing the relationships between units of dimensions. In a simple two dimensional example, one dimension is for two classes—a singly-linked list and a doubly-linked list—and another dimension is for additional operations—insert and delete operations. Cartesian combination of the classes and operations gives four possible programs (a two by two table). Units can be "folded" along each dimension: if the operation dimension is folded, there are two available classes, each with insert and delete operations. The class dimension can also be folded in the same way.

Batory *et al.* make two important claims about this approach: (1) it prevents possible invalid combinations of elements; for instance, it does not permit the selection of a doubly-linked list with operations for singly-linked, because folding always has to happen between rows or columns, and not between cells, (2) complexity of n dimensions can be reduced to the complexity of one dimension by folding them.

D. OVM [10]

Pohl *et al.* [10] differentiate between *variability in time*—denoting changes to artefacts over time—and *variability in space*—denoting static variability of artefacts. They also talk about *external variability*, those relevant to customers, and *internal variability*, those relevant to developers.

E. Staged Configuration [11]

Czarnecki *et al.* [11] propose a suite of "staged configuration" approaches where FDs are specialised in a stepwise fashion, and instantiated according to the stakeholder interests at each development stage [16].

With *specialisation* Czarnecki *et al.* refer to a process in which variabilities in FDs are removed. In other words, a more specialised FD has fewer variabilities than its parent FD. A fully specialised FD has no variability. A configuration, on the other hand, is an *instantiation* of an FD.

With *multi-level* staged configuration, Czarnecki *et al.* refer to a sequential process in which an FD is configured and specialised alternately by stakeholders in the development stages. For instance, a stakeholder will instantiate an FD by selecting features that are relevant to its requirements. The instance of the model, called a *configuration*, is then used to specialise the FD by removing parts of the model that are no longer available. The resulting FD is then instantiated by another stakeholder, and so the process repeats itself.

F. Reiser and Weber [1]

Reiser and Weber [1] consider the issue of managing large FDs and changes made to them over time, in the context of automotive software development. They point out that FDs need to reflect the structure of several organisations involved in the software development. They argue that dividing the FDs along organisational boundaries will make it difficult to propagate changes made to a local diagram. Managing a

Approach	Concerns	How concerns are addressed	Degree of formality	Degree of automation
FODA [7]	Feature relationships, such as standard, alternative/optional, specialisation, mutually exclusive and so on; dependency time, such as compile-time, load-time and run-time	Many of these concerns can be expressed in FDs. In a sense, these are the core concerns of FDs. However, exactly how these concerns are to be separated in FDs is not detailed in this technical report.	Diagrammatic only.	Not discussed in the report.
FORM [8], [15]	There are four main feature perspectives or concerns recognised here. They are: Capabilities (functional and non-functional), Operating Environments, Domain Technologies and Implementation Technologies.	Separation of these concerns are achieved by layering of the FDs according these concerns. For instance, FDs in the top layer address only the requirements of end users. Links between elements of the diagrams are expressed informally.	Diagrammatic only.	Not discussed in the report.
Batory *et at.* [9]	Concerns represented as dimensions. No complete list of dimensions are given. As examples, classes and class operations are used as dimensions.	Each concern is represented as an axis in an origami matrix. They propose a way of folding the matrix in order to generate valid configurations while managing the complexity.	Tool-support suggests a formal basis.	A set of tools is presented.
OVM [10]	Temporal variability versus design variability, and external variability versus internal variability	It is not clear how separation of concerns are reflected in the FDs.	Diagrammatic only.	A tool for OVM diagramming.
Staged Configuration [11]	Concerns are related to stakeholders in the development, such as the end user, developer, customer, and so on. No complete list is given.	Concerns are addressed through alternate specialisation and configuration of the FD.	Meta-model is given. Classen *et al.* [16] gives a semantics.	Not clear.
Reiser and Weber [1]	Organisational structure, changes to FDs	Hierarchical structuring of FDs where changes can be localised and propagated in a controlled way.	Diagrammatic only.	Diagramming tool.
Metzger *et at.* [12]	Product line variability vs software variability	Each of the two main concerns is expressed as an FD, and are then related through logical constraints. They describe a way of merging the FDs, and various automated analysis that can be performed.	FD semantics formalised.	Proof-of-concept prototype.
Hubaux *et at.* [13]	Design and runtime perspectives, default configuration	As it is an experience report, no approach for achieving the separation of concerns has been proposed.	Informal discussions.	Not applicable.
Grünbacher *et al.* [2]	Solution structure, multiple product lines (system of systems architecture), asset types, organisational structure, and market needs	Business and technical decisions are elicited. Variation points are detected using a tool, before model fragments are created.	Largely informal.	The DOPLER tool suite.
Tun *et at.* [14]	Requirements, problem world context, specification, quantitative constraints	FDs are divided into requirements, problem world context and specification FDs, whilst quantitative constraints are expressed over each of them. As in [12], they link the FDs through logical constraints.	Partial formalisation of the approach.	The use of an industrial tool illustrated.

Table I
SUMMARY OF CONCERNS

large global FD is also unsatisfactory because it will make FDs unmanageable.

They propose *multi-level feature trees* in which FDs are refined in a hierarchical fashion. Elements of a child FD can selectively reuse elements in the parent FD, allowing local changes to be made without affecting the global structure of the FDs.

G. *Metzger* et al. *[12]*

Metzger *et al.* propose distinguishing two kinds of variability, *product line variability* and *software variability*, where the former is concerned with "ability of a software system or artefact to be efficiently extended, changed, customized or configured for use in a particular context" [17], whilst the latter is concerned with "the variation between the systems that belong to a PL in terms of properties and qualities, like features that are provided or requirements that are fulfilled" [12].

As a simple example, they describe an on-line store where the software variability has an addition optional feature of *credit card payment*, and the *debit card payment* and *payment upon invoice* are alternative features.

H. *Hubaux* et at. *[13]*

Hubaux *et at.* investigate the practical challenges of applying FD languages. One of the challenges reported in this paper is that of making modelling perspectives (such as design time versus runtime perspectives) explicit in FDs. Another challenge is that of expressing default configurations for parts of the FDs.

I. *Grünbacher* et al. *[2]*

Grünbacher *et al.* discuss the challenges of structuring the modelling space for software product lines. They argue that maintaining a single FD for the entire system is not feasible and proceed to suggest strategies for feature modelling from various perspectives. They also present some examples of how these strategies can be applied, supported by existing tools.

J. Tun et at. *[14]*

Tun *et at.* propose separating the concerns of FDs into descriptions of *requirements*, *problem world context* and *specification* features, following the Jackson–Zave framework for requirements engineering [18], [19]. In addition, they express quantitative constraints on the feature modes, links connecting the three models, in order to generate feature configurations that satisfy stated requirements and quantitative constraints.

III. Discussions and Conclusions

Table I summarises the concerns recognised by the surveyed approaches and how the concerns are separated in those approaches. Notice that we are concerned with identifying concerns discussed by the surveyed approaches, rather than comparing the approaches on the basis of concerns they address.

It is interesting to note that the earlier FD languages (such as FODA [7] and FORM [8], [15]) seem to be more concerned with design and implementation issues, whilst later FD languages (such as Staged Configuration [11] and Feature Tree [1]) are more concerned with stakeholders and organisational structures. There is a tendency to expand the scope of FDs: in addition to describing variability in the design, a need for describing the variability in the wider system context has been recognised by FD approaches. This perhaps explains, in part, the phenomenon of increasing size and complexity of FDs.

The list of concerns recognised by the surveyed approaches, in particular by Lee *et al.* [8], [15], is very comprehensive. They range from cost of features, CPU platform to organisation structure. This indicates that variability has to be addressed at different times in the development and in different parts of the system structures.

Although, several concerns of FDs are well-known, there is no consensus on how best to separate these concerns. Many of the proposed approached are well-grounded and probably constructive when applied to real problems. More evidence of how they have been applied will strengthen confidence in these approaches.

We see a deep synergy between SPL and requirements engineering research. For instance, techniques on viewpoints [20], model synthesis [21], inconsistency management [22] may shed new lights on how concerns of FDs should be managed.

Despite the apparent difficulties, SPL engineers in various industries have been successfully producing commercial software used by many customers. Insightful reports on how SPL engineers actually manage concerns in FDs will have positive influence on the research.

Acknowledgement

We thank our colleagues Andreas Classen and Arnaud Hubaux for useful comments on an earlier draft. This research is supported by the CERUNA programme of the University of Namur, and by the Interuniversity Attraction Poles (IAP) Programme of the Belgian State, Belgian Science Policy.

References

[1] M.-O. Reiser and M. Weber, "Managing highly complex product families with multi-level feature trees," in *IEEE International Conference on Requirements Engineering (RE'06)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 146–155.

[2] P. Grünbacher, R. Rabiser, D. Dhungana, and M. Lehofer, "Structuring the product line modeling space: Strategies and examples," in *VaMoS*, ser. ICB Research Report, D. Benavides, A. Metzger, and U. W. Eisenecker, Eds., vol. 29. Universität Duisburg-Essen, 2009, pp. 77–82.

[3] E. W. Dijkstra, "On the role of scientific thought," in *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982, pp. 60–66.

[4] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. S. Jr., "*N* degrees of separation: Multi-dimensional separation of concerns," in *ICSE*, 1999, pp. 107–119.

[5] C. W. Krueger, "Using separation of concerns to simplify software product family engineering," in *Dagstuhl Seminar No. 01161*, 2001.

[6] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[7] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," SEI, Carnegie Mellon University, Tech. Rep. CMU/SEI-90-TR-21, November 1990.

[8] K. Lee, K. C. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering," in *Proceedings of the 7th International Conference on Software Reuse*. London, UK: Springer-Verlag, 2002, pp. 62–77.

[9] D. Batory, J. Liu, and J. N. Sarvela, "Refinements and multi-dimensional separation of concerns," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 5, pp. 48–57, 2003.

[10] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. NJ, USA: Springer, 2005.

[11] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged configuration through specialization and multi-level configuration of feature models," *Software Process: Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005.

[12] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, and G. Saval, "Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis," in *RE'07*, New Delhi, India, October 2007, pp. 243–253.

[13] A. Hubaux, P. Heymans, and D. Benavides, "Variability modelling challenges from the trenches of an open source product line re-engineering project," in *Proceedings of 12th International Software Product Lone Conference*. IEEE Computer Society, 2008, pp. 55–64.

[14] T. T. Tun, Q. Boucher, A. Classen, A. Hubaux, and P. Heymans, "Relating requirements and feature configurations: A systematic approach," in *Proceedings of International Software Product Line Conference (to appear)*, San Francisco, CA, USA, 24-28 August 2009.

[15] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, no. 0, pp. 143–168, 1998.

[16] A. Classen, A. Hubaux, and P. Heymans, "A formal semantics for multi-level staged configuration," in *VaMoS*, ser. ICB Research Report, D. Benavides, A. Metzger, and U. W. Eisenecker, Eds., vol. 29. Universität Duisburg-Essen, 2009, pp. 51–60.

[17] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques: Research articles," *Softw. Pract. Exper.*, vol. 35, no. 8, pp. 705–754, 2005.

[18] M. Jackson, *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. ACM Press, 1995.

[19] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM TOSEM*, vol. 6, no. 1, pp. 1–30, 1997.

[20] S. M. Easterbrook and B. A. Nuseibeh, "Using viewpoints for inconsistency management," *Software Engineering Journal*, vol. 11, no. 1, 1996.

[21] S. Uchitel and M. Chechik, "Merging partial behavioural models," in *ACM International Symposium on Foundations of Software Engineering (FSE'04)*, Newport Beach, 2004.

[22] G. Spanoudakis and A. Zisman, "Inconsistency management in software engineering: Survey and open research issues," in *Handbook of Software Engineering and Knowledge Engineering*, K. S. Chang, Ed. World Scientific Publishing Co, 2001, pp. 329–380.

[23] D. Benavides, A. Metzger, and U. W. Eisenecker, Eds., *Third International Workshop on Variability Modelling of Software-Intensive Systems, Seville, Spain, January 28-30, 2009. Proceedings*, ser. ICB Research Report, vol. 29. Universität Duisburg-Essen, 2009.