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; Sepa-
ration of Concerns; Survey;

I. INTRODUCTION

A Feature Diagram (FD) shows possible valid configura-
tions 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 tech-
niques? 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 al. [13], and Tun et al. [14].

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

A. FODA [7]

Kang et al. proposes dividing features into *standard fea-
tures, 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).
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. For instance, (object-oriented) classification is regarded as a dimension and classes of a software are units.

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 operations (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
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 al. [13]**

Hubaux *et al.* 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.
Tun et al. 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 approaches 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