Science of Computer Programming 76 (2011) 1130–1143

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A text-based approach to feature modelling: Syntax and semantics of TVL

Andreas Classen *,[1], Quentin Boucher, Patrick Heymans

PReCISE Research Centre, University of Namur, Rue Grandgagnage 21, B-5000 Namur, Belgium

ARTICLE INFO

Article history:
Received 5 March 2010
Received in revised form 5 August 2010
Accepted 21 October 2010
Available online 18 November 2010

Keywords:
Feature models
Code
Modelling
Language
Syntax
Semantics
Software product lines

ABSTRACT

In the scientific community, feature models are the *de-facto* standard for representing variability in software product line engineering. This is different from industrial settings where they appear to be used much less frequently. We and other authors found that in a number of cases, they lack concision, naturalness and expressiveness. This is confirmed by industrial experience.

When modelling variability, an efficient tool for making models intuitive and concise are feature attributes. Yet, the semantics of feature models with attributes is not well understood and most existing notations do not support them at all. Furthermore, the graphical nature of feature models' syntax also appears to be a barrier to industrial adoption, both psychological and rational. Existing tool support for graphical feature models is lacking or inadequate, and inferior in many regards to tool support for text-based formats.

To overcome these shortcomings, we designed TVL, a text-based feature modelling language. In terms of expressiveness, TVL subsumes most existing dialects. The main goal of designing TVL was to provide engineers with a human-readable language with a rich syntax to make modelling easy and models natural, but also with a formal semantics to avoid ambiguity and allow powerful automation.

1. Introduction

Software product line engineering (SPLE) is an increasingly popular software engineering paradigm which advocates systematic reuse across the software lifecycle. Central to the SPLE paradigm is the modelling and management of *variability*, i.e. *"the commonalities and differences in the applications in terms of requirements, architecture, components, and test artifacts"* [1]. Variability is typically expressed in terms of *features*, i.e. first-class abstractions that shape the reasoning of the engineers and other stakeholders [2]. Commercial (print on demand) printers, for instance, are developed as product lines and come with a broad range of features, such as support for spine captions, punching, or stapling. PRISMAprepare,[2] a commercial tool to prepare jobs for such printers, will serve as the running example in this paper.

A set of features can be seen as the specification of a particular product of the product line (PL). Feature models (FMs) [3,4] delimit the set of valid products of the PL. FMs are directed acyclic graphs, generally trees, whose nodes denote features and whose edges represent top-down hierarchical decomposition of features. The meaning of a decomposition link is that, if the parent feature is part of a product, then *some* of its child features have to be part of the product as well. Exactly which and how many of the child features have to be part depends on the type of the decomposition link. An excerpt of PRISMAprepare's FM is shown in Fig. 1, using the traditional graphical representation. The *and*-decomposition of the features *Document* and

* Corresponding author.
E-mail addresses: acs@info.fundp.ac.be (A. Classen), qbo@info.fundp.ac.be (Q. Boucher), phe@info.fundp.ac.be (P. Heymans).

[1] FNRS research fellow.

[2] See http://global.oce.com/products/prisma-prepare.

0167-6423/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2010.10.005

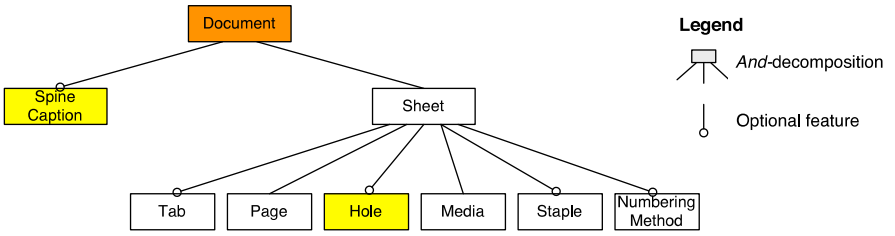

Fig. 1. Excerpt of PRISMAprepare's FM.

Sheet means that all their child features have to be selected, except for those that are optional, which is indicated by a hollow circle.

In the scientific community, FMs are the *de-facto* standard for representing the variability of an SPL. Several sources – our industry partners, discussions at the 2010 variability modelling (VaMoS) workshop [5] as well as recent literature reviews [6,7] – suggest that in the industrial world, in contrast, FMs appear to be used rarely.

One reason for this, we believe, is their lack of conciseness and naturalness when it comes to modelling realistic SPLs. Various industrial experiences have shown us that one of the most efficient tools for making FMs intuitive and concise are feature attributes [8], that is, typed parameters attached to features similar to attributes attached to classes in UML class diagrams. In the case of PRISMAprepare, for instance, the use of attributes reduced the number of features from 152 to 14, in another case one from 189 to 59 [9]. In the excerpt shown in Fig. 1, the colour of a feature indicates the number of attributes (yellow meaning *one* and orange meaning *two*) that the feature has. Discussions with engineers also showed that the resulting diagrams are more natural and easier to understand [9]. For instance, without attributes, we are forced to model alternative choices that are not further decomposed as a *xor*-decomposed feature group. A more concise solution would be to use an enumerated attribute. The semantics is exactly the same, but the notation is much more concise. In spite of all that, the semantics of FMs with attributes is not well understood and most existing notations and tools do not support them at all.

Another likely reason for the difficulty of using FMs in practice is the graphical nature of their syntax. Almost all existing FM languages are based on the FODA notation [3] which uses graphs with nodes and edges in a 2D space, as shown in Fig. 1. Feature attributes, to begin with, are intrinsically textual in nature and do not easily fit into this representation. Furthermore, *constraints* on the FM are often expressed as textual annotations using Boolean operators. If they were given a graphical syntax, attributes and constraints would only clutter a FM. When working with engineers, we also observed that a graphical syntax is a psychological barrier (having to draw models is deemed tedious and cumbersome by engineers) and poses a tooling problem. Existing tools for graphical FMs are generally research prototypes and are inferior in many regards to tool support for text-based formats (viz. text editors, source control systems, diff tools, no opaque file formats and so on).

To overcome these shortcomings, we designed `TVL` (Textual Variability Language), a text-based FM language. The idea of using text to represent variability in SPLE is not new [10,11] but seems to be recently gaining popularity [12,13]. In terms of expressiveness, `TVL` subsumes most existing dialects. The main goal of designing `TVL` was to provide engineers with a human-readable language with a rich syntax to make modelling easy and models natural, but also with a formal semantics to avoid ambiguity and allow powerful automation. Further goals for `TVL` were to be *lightweight* (in contrast to the verbosity of XML for instance) and to be *scalable* by offering mechanisms for structuring the FM in various ways.

Keeping with the tradition of the authors [4,14], `TVL` is defined formally. Its concrete, C-like, syntax is described by an LALR grammar, but it also has a mathematical abstract syntax and a denotational semantics. Having a well-defined tool-independent semantics further distinguishes `TVL` from most existing languages. A formal semantics is crucial for languages that are to be widely used or to serve as a format for information exchange. Moreover, it allows anyone to implement the language, serving as specification and reference. `TVL` is a *pure* language in the sense that all its constructs directly have a precise interpretation in the formal semantics. A reference implementation including a parser and a reasoning library is available online.[3]

The remainder of the paper is structured as follows. We survey related work in Section 2. Section 3 introduces `TVL` with code snippets from our running example. Section 4 gives well-formedness rules for `TVL` models while Section 5 specifies the formal semantics. We evaluate `TVL` in Section 6, followed by a description of our implementation in Section 7 and conclude in Section 8.

2. Related work

In the literature, graphical FM notations based on FODA [3] are by far the most widely used. Most of the subsequent proposals such as FeatuRSEB [15], FORM [16] or Generative Programming [17] are only slightly different from the original graphical syntax (e.g. by adding boxes around feature names).

But a number of textual FM languages were also proposed in the literature. Table 1 compares them. The criteria are (i) *human readability*, i.e. whether the language is meant to be read and written by a human; (ii) support for attributes;

[3] http://www.info.fundp.ac.be/~acs/tvl.