Information and Software Technology 55 (2013) 1925–1947

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

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

A formal framework for software product lines ☆

César Andrés, Carlos Camacho, Luis Llana *

Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Madrid, Spain

ARTICLE INFO

Article history:
Received 24 May 2012
Received in revised form 19 May 2013
Accepted 20 May 2013
Available online 31 May 2013

Keywords:
Formal methods
Software product lines
Feature models

ABSTRACT

Context: A Software Product Line is a set of software systems that are built from a common set of features. These systems are developed in a prescribed way and they can be adapted to fit the needs of customers. Feature models specify the properties of the systems that are meaningful to customers. A semantics that models the feature level has the potential to support the automatic analysis of entire software product lines.
Objective: The objective of this paper is to define a formal framework for Software Product Lines. This framework needs to be general enough to provide a formal semantics for existing frameworks like FODA (Feature Oriented Domain Analysis), but also to be easily adaptable to new problems.
Method: We define an algebraic language, called SPLA, to describe Software Product Lines. We provide the semantics for the algebra in three different ways. The approach followed to give the semantics is inspired by the semantics of process algebras. First we define an operational semantics, next a denotational semantics, and finally an axiomatic semantics. We also have defined a representation of the algebra into propositional logic.
Results: We prove that the three semantics are equivalent. We also show how FODA diagrams can be automatically translated into SPLA. Furthermore, we have developed our tool, called AT, that implements the formal framework presented in this paper. This tool uses a SAT-solver to check the satisfiability of an SPL.
Conclusion: This paper defines a general formal framework for software product lines. We have defined three different semantics that are equivalent; this means that depending on the context we can choose the most convenient approach: operational, denotational or axiomatic. The framework is flexible enough because it is closely related to process algebras. Process algebras are a well-known paradigm for which many extensions have been defined.

1. Introduction

Software Product Lines [1,2], in short SPLs, constitute a paradigm for which industrial production techniques are adapted and applied to software development. In contrast to classical techniques, where each company develops its own software product, SPLs define generic software products, enabling mass customization [3]. Generally speaking, using SPLs provide a systematic and disciplined approach to developing software. It covers all aspects of the software production cycle and requires expertise in data management, design, algorithm paradigms, programming languages, and human–computer interfaces.

When developing SPLs, it is necessary to apply sound engineering principles in order to obtain economically reliable and efficient software. *Formal methods* [4–9] are useful for this task. A formal method is a set of mathematical techniques that allows automated design, specification, development and verification of software systems. For this process to work properly, a well defined formalism must exist. There are many formalisms to represent SPLs [10–13]. We focus on one of the most widely approaches: *Feature models* [10,13].

A feature model is a compact representation of all the products of an SPL in terms of commonality and variability. Generally these features are related using a tree-like diagram. A variation point is a place where a decision can be made to determine if none, one, or more features can be selected to be part of the final product. For instance, in these models we can represent the following property:

There exists a product with features A *and* C.

In addition, it is easy to represent constraints over the features in feature models. For instance, we could represent the following property:

In any valid product, if feature C *is included then features* A *and* B *must also be included.*

☆ Research partially supported by the Spanish MEC project TIN2009-14312-C02-01 and TIN2012-36812-C02-01.

* Corresponding author. Tel.: +34 913944527.
E-mail addresses: rasec.andres@gmail.com (C. Andrés), carloscamachoucv@gmail.com (C. Camacho), llana@ucm.es (L. Llana).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.05.005

Feature Oriented Domain Analysis [10], in short FODA, is a feature model to represent SPLs. This model allows us to graphically represent *features* and their *relationships*, in order to define *products* in an SPL. The graphical structure of a FODA model is represented by a FODA *Diagram*. A FODA Diagram is essentially an intuitive and easy to understand graph where there is relevant information about the features. This diagram has two different elements: the set of nodes and the set of arcs. The former represents the features of the SPL. The latter represents the relationships and the constraints of the SPL. We introduce the basic components of a FODA diagram in Fig. 1. With these elements we can model complex SPLs.

For instance, let us look at the FODA Diagrams in Fig. 2. The Examples **a** and **b** show two SPLs with possibly two possible features A and B. With respect to **a**, the feature A will appear in all valid products of this SPL, while B is optional. Therefore, the valid set of products of this FODA Diagram is one product with A and one product with features A and B. In **b** both features are mandatory, i.e. any product generated from this SPL will contain features A and B.

Example **c**, represents an SPL with a *choose-one* operator. There are three different features: A, B and C in this diagram. Any valid product of **c** will contain A and one of these features B or C. The *conjunction* operator is shown in **d** and **e**. In both examples two branches leave feature A. On the one hand, the branches in Example **d** are mandatory. This means that there is only one product derived from this diagram: the one that contains features A, B, and C. On the other hand, one branch in Example **e** is optional and the other is mandatory. This means that there are two products derived from this diagram: one with features A and C, and one with features A, B, and C.

Finally, more complex properties appear in Examples **f** and **g**. In these diagrams there are tree constraints combined with optional features. In **f**, there is an *exclusion* constraint: If B is included in a product then feature C cannot appear in the same product. In **g**, there is a *require* constraint: If B is included then C must also be included.

Although FODA Diagrams are very intuitive, sometimes it can be hard to analyze all the restrictions and the relationships between features. In order to make a formal analysis we have to provide a *formal semantics* for the diagrams. To obtain a formal semantics for SPLs, first we need a formal language. In this paper we define a formal language called SPLA. As we will see in Section 3, all FODA Diagrams can be automatically translated into SPLA.

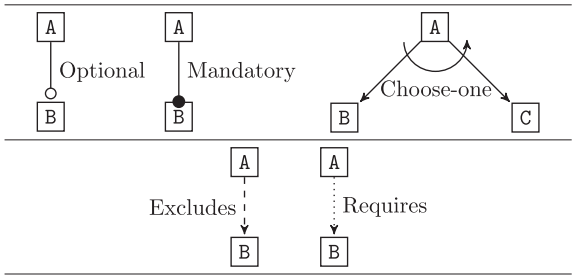

Fig. 1. FODA diagram representation.

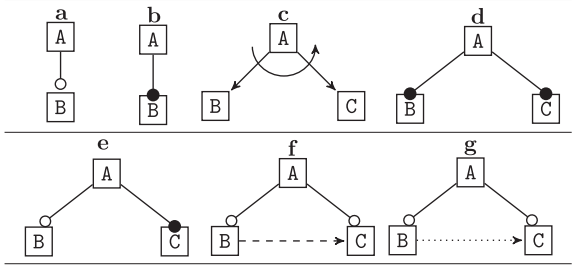

Fig. 2. Examples of FODA diagrams.

After presenting SPLA, we need to introduce the formal semantics of this algebra. There is previous work on formalizing FODA and feature models [30,22,31–33,23,24] that we briefly review in the next section. The approach we follow in this paper is inspired by classical process algebras [4,6,5]. We define three different semantics for the language. First we introduce an operational semantics whose computations give the products of an SPL. Next we define a denotational semantics that is less intuitive but easier to implement. Finally we have defined an axiomatic semantics. We prove that all three semantics are equivalent to each other.

In addition to presenting the formal framework, we have developed a tool called AT. This tool is an implementation of the formal semantics presented in this paper. Using AT it is possible to check properties such as:

- *Can this SPL produce a valid product?*
- *How many valid products can we build within this SPL?*
- *Given an SPL diagram, can we generate an equivalent SPL diagram with fewer restrictions than the first one?*

This tool is completely implemented in JAVA. This tool has a module to check the satisfiability of an SPL diagram. We carry out some experiment using diagrams obtained from the random SPL diagram generator Betty. These experiments shows the scalability of AT. We have checked satisfiability of diagrams with 13,000 features; such diagrams are relatively large given the state of the art.

In this paper we present a syntactic and semantic framework that formalizes FODA-like diagrams. First we present a syntax with the basic operators presented in a FODA-like diagram. Next we define an operational semantics. This semantics is intuitive and it captures the notions of the operators. After the operational semantics, we give the denotational semantics. This semantics is more appropriate for obtaining the products of an SPL. We prove that both semantics are equivalent. We also define an axiomatic semantics. As far as we know, this semantics does not appear in any of the previous frameworks. We prove that this semantics is sound and complete with respect to the previous ones. Inspired by recent works in process algebras, we also give a way to represent the terms of the algebra into propositional logic. Finally we have implemented a tool that supports our framework. The tool is split into two modules. One module deals with the denotational and axiomatic semantics while the other deals with the representation into propositional logic. The second module uses SAT-solver to check the satisfiability of an SPL.

This paper tries to show that SPLs can benefit from the process algebra community mainly because process algebras have been studied from many points of view. For instance, there have been numerous proposals to incorporate non-functional aspects such as time and probabilities. In particular, we are currently working in the following aspects. First, we are studying how to introduce, the notions of costs and time. In this context, it is also important the order in which products are elaborated, and therefore, sequences instead of sets have to be used. Second, we would like to work with models that indicate the probability of a product. This can be applied, for example, in software testing, so that we can add more resources to test the products with higher probabilities. Moreover, our semantic approach, based on alternative semantics that are shown to be equivalent, can be used in further extensions of both the formalism used in this paper and other formalisms of similar nature.