Reverse engineering language product lines from existing DSL variants

David Méndez-Acuña*, José A. Galindo, Benoît Comemiale, Arnaud Blouin, Benoît Baudry

INRIA/IRISA and University of Rennes 1 Campus de Beaulieu, Rennes, France

ARTICLE INFO

Article history:
Received 3 October 2016
Revised 30 March 2017
Accepted 12 May 2017
Available online xxx

Keywords:
Language product lines
Software languages engineering
Domain-specific languages
Reverse-engineering

ABSTRACT

The use of domain-specific languages (DSLs) has become a successful technique to develop complex systems. In this context, an emerging phenomenon is the existence of DSL variants, which are different versions of a DSL adapted to specific purposes but that still share commonalities. In such a case, the challenge for language designers is to reuse, as much as possible, previously defined language constructs to narrow implementation from scratch. To overcome this challenge, recent research in software languages engineering introduced the notion of language product lines. Similarly to software product lines, language product lines are often built from a set of existing DSL variants.

In this article, we propose a reverse-engineering technique to ease-off such a development scenario. Our approach receives a set of DSL variants which are used to automatically recover a language modular design and to synthesize the corresponding variability models. The validation is performed in a project involving industrial partners that required three different variants of a DSL for finite state machines. This validation shows that our approach is able to correctly identify commonalities and variability.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

The increasing complexity of modern software systems has motivated the need of raising the level of abstraction at which software is designed and implemented (Chechik et al., 2010). The use of domain-specific languages (DSLs) has emerged in response to this need as an alternative to express software solutions in relevant domain concepts, thus favoring separation of concerns and hiding fine-grained implementation details (Jézéquel et al., 2015b). DSLs are software languages whose expressiveness is focused on a well-defined domain and which provide abstractions a.k.a., language constructs that address a specific purpose (Mernik et al., 2005). The adoption of such a language-oriented vision has motivated the construction of a large variety of DSLs. There are, for example, DSLs to build graphical user interfaces (Oney et al., 2012), to specify security policies (Lodderstedt et al., 2002), or to ease off mobile applications’ prototyping (Ribeiro and da Silva, 2014).

Despite all the advantages furnished by DSLs in terms of abstraction and separation of concerns, this approach has also important drawbacks that put into question its benefits (Gray et al., 2008). One of those drawbacks is associated to the elevated costs of the language development process. The construction of DSLs is a time consuming activity that requires specialized background (Jézéquel et al., 2015b); language designers must own solid modeling skills and technical knowledge to conduct the definition of complex artifacts such as metamodels, grammars, interpreters, or compilers (Jézéquel et al., 2015b).

The development of DSLs becomes more complex when we consider that DSLs often have many variants. A variant is a new version of a given DSL that introduces certain differences in terms of syntax and/or semantics (Homer et al., 2014). Typically, language variants appear under two situations. The first situation is the use of well-known formalisms through different domains. Consider the case of finite state machines, which have been used in the construction of DSLs for a large spectrum of domains such as definition of graphical user interfaces (Oney et al., 2012) or games prototyping (Funk and Rauterberg, 2012). Those DSLs share typical state machine concepts such as states or transitions. However, each DSL adapts those abstractions to address the particularities of its domain.

The second situation that favors the existence of DSL variants is when the complexity of a given domain requires the construction of several DSLs with different purposes. In such a case, the domain abstractions of the DSLs are similar, but their concrete implementations require adaptations. For instance, suppose two DSLs: the former is a DSL for specification and verification of railway schemes (James and Roggenbach, 2014); the latter is a DSL for modeling and reasoning on railway systems’ capacity and security (Iliasov et al., 2013). These DSLs share certain domain abstractions i.e., railway management. However, they both require different semantics and specialized constructs to achieve their purposes.

* Corresponding author.
E-mail addresses: damenac@gmail.com (D. Méndez-Acuña), jgalindo@inria.fr (J.A. Galindo), benoît.comemiale@inria.fr (B. Comemiale), arnaud.blouin@inria.fr (A. Blouin), benoit.baudry@inria.fr (B. Baudry).

http://dx.doi.org/10.1016/j.jss.2017.05.042
0164-1212/© 2017 Elsevier Inc. All rights reserved.

Please cite this article as: D. Méndez-Acuña et al., Reverse engineering language product lines from existing DSL variants, The Journal of Systems and Software (2017), http://dx.doi.org/10.1016/j.jss.2017.05.042
The phenomenon of DSL variants is not a problem itself but reflects the abstraction power of certain well-known formalisms—such as state machines or petri nets—that, with proper adaptations, can fit various domains. Besides, it shows how different issues in a same domain can be addressed by diverse and complementary DSLs. Nevertheless, when the same team of language designers has to deal not only with the construction of DSLs but also with the definition of several variants, then their work becomes even more challenging. After all, at implementation level each DSL variant is a complete language itself requiring tooling such as editors, interpreters, compilers, and so on.

In this context, the challenge for language designers is to take advantage of the commonalities existing among DSL variants by reusing, as much as possible, formerly defined language constructs (Zschaler et al., 2010). The objective is to leverage previous engineering efforts to minimize implementation from scratch. To achieve such a challenge, the research community in software language engineering has proposed the use of Software Product Line Engineering (SPLE) in the construction of DSLs (White et al., 2009; Méndez-Acuña et al., 2016b). This led to the notion of Language Product Line Engineering (LPLE), i.e., the construction of software product lines where the products are languages (Zschaler et al., 2010; Kühn et al., 2015).

Similarly to software product lines, language product lines can be built from a set of existing DSL variants through reverse-engineering techniques (Kühn and Cazzola, 2016). Those techniques should provide mechanisms for: (1) recovering of a language modular design including all the language constructs existing in the DSL variant; and (2) synthesis of the corresponding variability models.

In a previous work (Méndez-Acuña et al., 2016a; 2016c), we introduced an approach to automatically infer a language modular design from a given set of DSL variants. In this article, we extend that work to provide a complete reverse-engineering technique that produces not only the language modular designs, but the entire language product line. In that sense, the delta of this article with respect to the previous one is the synthesis of the variability models. Those variability models are specified in terms of well-known formalisms i.e., feature models (FM) and orthogonal variability models (OVM) in such a way that they encode the variability of a language product line in a compact way while considering the diverse dimensions that such a variability may present. We also show how those variability models can be used to configure and assembly new DSL variants.

We validate our approach through the implementation within an industrial project, which is composed of three variants of a DSL for finite-state machines (Crate and Dingel, 2007). In this project we manually developed an oracle to know in advance the existing variation points. Then, we execute our approach on these DSL variants and we compare the produced results against the expected ones. The result of this comparison shows that our reverse engineering technique is correct since all the detected variation points correspond to real differences in the DSL variants. Also, this validation allowed us to identify certain threats to validity regarding the level of granularity of the detected variation points.

The remainder of this article is structured as follows: Section 2 describes the problem statement. Section 3 introduces our approach. Section 4 presents the experiments executed in the context of the VaryMDE project that are used as a validation. Section 5 discusses the related work. Finally, Section 6 concludes the article.

2. Problem statement

Similarly to software product lines (Linden et al., 2007), the development process of language product lines can be divided into two phases: domain engineering and application engineering. During the domain engineering phase, language designers build the language product line. This process includes the design and implementation of a set of interdependent language modules that implement the language features and the construction of variability models encoding the rules in which those features can be combined to produce valid DSL variants. During the application engineering phase, diverse DSL variants are derived according to specific needs. Such a derivation process comprises the selection of the features to include in a given DSL variant, i.e., language configuration and the assembly of the corresponding language modules, i.e., language modules composition.

Note, however, that in bottom-up language product lines, application engineering is performed first and domain engineering is performed afterwards through reverse-engineering techniques. During the application engineering phase, language designers build an initial DSL. As the language development project evolves, some DSL variants are needed to address new requirements. Language designers create new development branches where they implement these new variants with the corresponding adaptations. At some point, language designers realize that there is potential reuse among the variants. Hence, they launch a reverse engineering process—which in this case corresponds to the domain (re)engineering phase—where the existing DSL variants are used to build up a language product line. Using this language product line, language designers can revisit the application engineering phase in order to create new DSL variants.

2.1. Motivating scenario

Suppose a team of language designers working on the construction of the DSL for finite state machines. Initially, language designers followed the UML specification (OMG) to define language constructs such as states, regions, transitions, and triggers. Those language constructs are specified in terms of their syntax and semantics. So, at the end of the language development process, language designers release an executable DSL whose behavior complies the UML specification.

Once this first DSL was released, the language designers are asked to build a new variant which must comply the Rhapsody specification (Harel and Kugler, 2004) (i.e., another formalism to finite state machines). This new variant shares many commonalities with UML state machines, but introduces differences at both syntax and semantics levels (Crane and Dingel, 2007). After building this second variant, language designers obtained two different DSLs implementing different formalisms of state machines. Those DSL variants have some commonalities among them. And at the same time, the DSL have some particularities that make them unique.

Note that this process is repeated each time language designers have to build a new variant of the DSL for each new FSM formalism (e.g., Stateflow (Martaj and Mokhtari, 2010) or Harel state machines (Harel and Naamad, 1996)). This becomes specially challenging when final users need to combine some specifications to define hybrid formalisms. While several approaches have been proposed to reverse engineering software product lines from existing product variants (Lopez-Herrejon et al., 2015; Martinez et al., 2015a; 2015b), in this article we propose techniques to reverse engineering language product lines from existing DSL variants.

2.2. Scope: Executable Domain Specific Languages

All the ideas presented in this article are focused to executable domain specific modeling languages (xDSMLs) where the abstract syntax is specified through metamodels, and the dynamic semantics is specified operationally as a set of domain specific actions.
دریافت فوری متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سیستم هوشمند رهگیری سفارشات