Reverse Engineering Language Product Lines

David Méndez-Acuña, José A. Galindo, Benoit Combemale, and Benoit Baudry

University of Rennes 1, INRIA/IRISA. France

{david.mendez-acuna,jagalindo,benoit.combemale,benoit.baudry}@inria.fr

Abstract. The use of domain-specific languages (DSLs) is becoming a successful technique in the implementation of complex systems. However, the construction of this type of languages is time-consuming and requires highly-specialized knowledge and skills. Hence, researchers are currently seeking approaches to leverage reuse during the DSLs development in order to minimize implementation from scratch. An important step towards achieving this objective is to identify commonalities among existing DSLs. These commonalities constitute potential reuse that can be exploited by using reverse-engineering methods. In this paper, we present an approach intended to identify sets of DSLs with potential reuse. We also provide a mechanism that allows language designers to measure such potential reuse in order to objectively evaluate whether it is enough to justify the applicability of a given reverse-engineering process. We validate our approach by evaluating a large amount of DSLs we take from public `GitHub` repositories.

1 Introduction

The use of domain-specific languages (DSLs) has become a successful technique to achieve separation of concerns in the development of complex systems [8]. A DSL is a software language in which expressiveness is scoped into a well-defined domain that offers a set of abstractions (a.k.a., language constructs) needed to describe certain aspect of the system [6]. For example, in the literature we can find DSLs for prototyping graphical user interfaces [22], specifying security policies [17], or performing data analysis [10].

Naturally, the adoption of such language-oriented vision relies on the availability of the DSLs needed for expressing all the aspects of the system under construction. Besides, typically every software project is unique and requires the construction of tailor-made DSLs for dealing with all its particular aspects [4]. This fact carries the development of many DSLs which is a challenging task due the specialized knowledge it demands. A language designer must own not only quite solid modeling skills but also the technical expertise for conducting the definition of specific artifacts such as grammars, metamodels, compilers, and interpreters. As a mater of fact, the ultimate value of DSLs has been severely limited by the cost of the associated tooling (i.e., editors, parsers, etc...) [13].

To improve cost-benefit when using DSLs, the research community in software languages engineering has proposed mechanisms to increase reuse during

the DSLs construction process. The idea is to leverage previous engineering efforts and minimize implementation from scratch [24]. These reuse mechanisms are based on the premise that "software languages are software too" [11] so it is possible to use software engineering techniques to facilitate their construction [14]. In particular, there are approaches that take ideas from Component-Based Software Engineering (CBSE) [5] and Software Product Lines Engineering (SPLE) [27] during the construction of new DSLs.

A classical way for adopting the aforementioned reuse mechanisms is to construct DSLs as building blocks (a.k.a, language modules) that can be later extended and/or imported as part of the specifications of future DSLs. Indeed, there are approaches that support definition of interdependent language modules that can be later composed among them [23,19,15]. However, the definition of language modules that can be useful in future DSLs is not easy. In part, this is due to the fact that, as the same as in the general case of software development processes, a language module is always designed to work in a particular context that, in general, depends on the current project.

A more pragmatical approach to leverage reuse in the construction of DSLs is to focus on legacy DSLs [9]. That is, to exploit reuse in existing DSLs that have been developed independently and without being designed to be reused but that share some commonalities (i.e., they provide similar language constructs). Using this strategy, language modules can be extracted from the commonalities of the DSLs by means of reverse-engineering methods. This type of *a posteriori* reuse permits not only to reduce maintenance cost but also to facilitate the development of new DSLs that can be built from the composition of the resulting language modules.

As the reader may guess, the very first step towards the application of this strategy is to identify potential reuse in a set of existing DSLs. Language designers need to detect commonalities between the DSLs under study and objectively evaluate whether those commonalities represent potential reuse enough to justify the effort associated to the reverse-engineering process. This first step is quite important because reverse-engineering is an expensive process. Language designers must be sure that applying it will actually improve cost-benefit.

In this paper, we present an approach that takes as input a set of DSLs and identifies the commonalities existing among them. To do so, we perform static analysis on the artifacts where the DSLs are specified and compare language constructs at the level of the syntax and semantics in order to detect commonalities. Besides, our approach computes a set of metrics that permit to objectively evaluate those commonalities. This second part of our approach is based on some reuse metrics already proposed in the literature for the general case of software development [1,2] that we adapt them to the specific case of DSLs development.

We validate our approach by taking as input an important amount of languages available on `GitHub` public repositories. The results of this validation are quite promising since they show that there is a large amount of sets of DSLs that share language constructs and where reuse opportunities are evident. All

the ideas presented in this paper are implemented in an Eclipse-based tool that can be downloaded and installed as well as the validation scenarios.

This paper is organized as follows: Section **??** introduces a set of preliminary definitions/assumptions that we use all along the paper. Section **??** describes our approach. Section **??** validates the approach on DSLs we take from GitHub. Section **??** discusses the threads to validity. Section **??** presents the related work and, finally, Section **??** concludes the paper.

Acknowledgments

The research presented in this paper is supported by the European Union within the FP7 Marie Curie Initial Training Network “RELATE" under grant agreement number 264840 and VaryMDE, a collaboration between Thales and INRIA.

References

1. C. Berger, H. Rendel, and B. Rumpe. Measuring the ability to form a product line from existing products. volume abs/1409.6583. 2014.
2. C. Berger, H. Rendel, B. Rumpe, C. Busse, T. Jablonski, and F. Wolf. Product Line Metrics for Legacy Software in Practice. In *SPLC 2010 : Proceedings of the 14th International Software Product Line Conference*, pages 247–250. Univ., Lancaster, 2010.
3. B. Biegel and S. Diehl. Jccd: A flexible and extensible api for implementing custom code clone detectors. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 167–168, New York, NY, USA, 2010. ACM.
4. T. Clark and B. Barn. Domain engineering for software tools. In I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, editors, *Domain Engineering*, pages 187–209. Springer Berlin Heidelberg, 2013.
5. T. Cleenewerck. Component-based dsl development. In F. Pfenning and Y. Smaragdakis, editors, *Generative Programming and Component Engineering*, volume 2830 of *Lecture Notes in Computer Science*, pages 245–264. Springer Berlin Heidelberg, 2003.
6. B. Combemale, J. Deantoni, B. Baudry, R. France, J.-M. Jézéquel, and J. Gray. Globalizing modeling languages. *Computer*, 47(6):68–71, June 2014.
7. B. Combemale, J. Deantoni, M. Vara Larsen, F. Mallet, O. Barais, B. Baudry, and R. France. Reifying concurrency for executable metamodeling. In M. Erwig, R. F. Paige, and E. Van Wyk, editors, *6th International Conference on Software Language Engineering*, volume 8225 of *SLE*, pages 365–384, Indianapolis, IN, United States, Oct 2013. Springer.
8. S. Cook. Separating concerns with domain specific languages. In D. Lightfoot and C. Szyperski, editors, *Modular Programming Languages*, volume 4228 of *Lecture Notes in Computer Science*, pages 1–3. Springer Berlin Heidelberg, 2006.
9. T. Degueule, B. Combemale, A. Blouin, O. Barais, and J.-M. Jézéquel. Melange: A meta-language for modular and reusable development of dsls. In *8th International Conference on Software Language Engineering (SLE)*, Pittsburgh, United States, Oct. 2015.

10. J. Eberius, M. Thiele, and W. Lehner. A domain-specific language for do-it-yourself analytical mashups. In A. Harth and N. Koch, editors, *Current Trends in Web Engineering*, volume 7059 of *Lecture Notes in Computer Science*, pages 337–341. Springer Berlin Heidelberg, 2012.
11. J.-M. Favre, D. Gasevic, R. L‰mmel, and E. Pek. Empirical language analysis in software linguistics. In *Software Language Engineering*, volume 6563 of *LNCS*, pages 316–326. Springer, 2011.
12. D. Harel and B. Rumpe. Meaningful modeling: what's the semantics of "semantics"? *Computer*, 37(10):64–72, Oct 2004.
13. J.-M. Jézéquel, D. Méndez-Acuña, T. Degueule, B. Combemale, and O. Barais. When Systems Engineering Meets Software Language Engineering. In *CSD&M'14 - Complex Systems Design & Management*, Paris, France, Nov. 2014. Springer.
14. A. Kleppe. The field of software language engineering. In D. Ga?evi?, R. L‰mmel, and E. Van Wyk, editors, *Software Language Engineering*, volume 5452 of *Lecture Notes in Computer Science*, pages 1–7. Springer Berlin Heidelberg, 2009.
15. H. Krahn, B. Rumpe, and S. Völkel. Monticore: a framework for compositional development of domain specific languages. *International Journal on Software Tools for Technology Transfer*, 12(5):353–372, 2010.
16. L. Lafi, S. Hammoudi, and J. Feki. Metamodel matching techniques in mda: Challenge, issues and comparison. In L. Bellatreche and F. Mota Pinto, editors, *Model and Data Engineering*, volume 6918 of *Lecture Notes in Computer Science*, pages 278–286. Springer Berlin Heidelberg, 2011.
17. T. Lodderstedt, D. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002 - The Unified Modeling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer Berlin Heidelberg, 2002.
18. D. Lucanu and V. Rusu. Program equivalence by circular reasoning. In E. Johnsen and L. Petre, editors, *Integrated Formal Methods*, volume 7940 of *Lecture Notes in Computer Science*, pages 362–377. Springer Berlin Heidelberg, 2013.
19. M. Mernik. An object-oriented approach to language compositions for software language engineering. *J. Syst. Softw.*, 86(9):2451–2464, Sept. 2013.
20. M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, Dec. 2005.
21. P. D. Mosses. The varieties of programming language semantics and their uses. In D. Bjørner, M. Broy, and A. V. Zamulin, editors, *Perspectives of System Informatics*, volume 2244 of *Lecture Notes in Computer Science*, pages 165–190. Springer Berlin Heidelberg, 2001.
22. S. Oney, B. Myers, and J. Brandt. Constraintjs: Programming interactive behaviors for the web by integrating constraints and states. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 229–238, New York, NY, USA, 2012. ACM.
23. E. Vacchi and W. Cazzola. Neverlang: A framework for feature-oriented language development. *Computer Languages, Systems & Structures*, 43:1 – 40, 2015.
24. T. van der Storm, W. Cook, and A. Loh. Object grammars. In K. Czarnecki and G. Hedin, editors, *Software Language Engineering*, volume 7745 of *Lecture Notes in Computer Science*, pages 4–23. Springer Berlin Heidelberg, 2013.
25. M. Völter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. L. Kats, E. Visser, and G. Wachsmuth. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013.

26. S. Zschaler, D. S. Kolovos, N. Drivalos, R. F. Paige, and A. Rashid. Domain-specific metamodelling languages for software language engineering. In M. van den Brand, D. Ga?evi?, and J. Gray, editors, *Software Language Engineering*, volume 5969 of *Lecture Notes in Computer Science*, pages 334–353. Springer Berlin Heidelberg, 2010.
27. S. Zschaler, P. Sánchez, J. a. Santos, M. Alférez, A. Rashid, L. Fuentes, A. Moreira, J. a. Araújo, and U. Kulesza. Vml* a family of languages for variability management in software product lines. In M. van den Brand, D. Ga?evi?, and J. Gray, editors, *Software Language Engineering*, volume 5969 of *Lecture Notes in Computer Science*, pages 82–102. Springer Berlin Heidelberg, 2010.