Identifying Potential Reuse in the Development of Domain-Specific Languages

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

University of Rennes 1, INRIA/IRISA. France
{david.mendez-acuna,jagalindo,benoit.combemale,arnaud.blouin,benoit.baudry}@inria.fr

Abstract. Domain-specific languages (DSLs) are 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 objectively evaluate whether potential reuse 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 approach to achieve separation of concerns in the development of complex systems [5]. A DSL is a software language in which expressiveness is scoped into a well-defined domain that offers the abstractions needed to describe certain aspect of the system [4]. For example, in the literature we can find DSLs for prototyping graphical user interfaces [13], specifying security policies [12], or performing data analysis [6].

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. This fact carries the development of these DSLs which is a challenging task also due to the specialized knowledge it requires. 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 matter of fact, the ultimate value of DSLs has been severely limited by the cost of the associated tooling (i.e., editors, parsers, compilers, etc...) [9].

To deal with such complexity, the research community in Software Languages Engineering (SLE) has proposed mechanisms to increase reuse within the construction of DSLs. The idea is to leverage previous engineering efforts and minimize implementation from scratch. These reuse mechanisms are based on the premise that "software languages are software too" [7] so it is possible to use software engineering techniques to facilitate their construction [10]. In particular, there are approaches that take ideas from Component-Based Software Engineering (CBSE) and Software Product Lines Engineering (SPLE) during the construction of new DSLs [11,16,18].

The aforementioned reuse mechanisms can be adopted by means of two different approaches: *top-down* and *bottom-up*. The top-down approach proposes the design and implementation from scratch of reusable language modules that can be employed in the construction of several DSLs. Differently, the bottom-up approach proposes to extract those language modules from existing DSLs that share some commonalities which can be properly encapsulated.

Whereas the major complexity in top-down approaches is that language designers should design language modules by trying to predict their potential reuse; bottom-up approaches do not have to deal with that problem. Rather, the extraction of the reusable language modules is based on the detection of commonalities in existing DSLs. Consequently, bottom-up approaches can be considered as promising approach and, indeed, there are already in the literature some works (e.g., [15]) advancing towards that direction.

It is worth noting that the very first step towards a reverse engineering process for DSLs is the identification of potential reuse. In other words, before applying reverse engineering, language designers need: (1) to identify sets of DSLs with potential reuse and; (2) be sure that the potential reuse is enough to justify the associated effort.

In this paper, we present a proposal to tackle this issue. Concretely, we introduce an approach that takes as input a set of DSLs and identifies which of them share commonalities so they have potential reuse. To do so, we perform static analysis on the artifacts where the DSLs are specified (i.e., metamodels and semantic definitions). Besides, our approach computes a set of metrics on the DSLs that permit to objectively evaluate if the potential reuse justifies the effort required by the reverse-engineering process. To do so, we take some reuse metrics already proposed in the literature for the general case of software development [1,2] and we adapt them to the specific case of DSLs development. The validation of our approach was performed by taking as input an important amount of languages available on GitHub public repositories. The results of this validation are quite promising.

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.

The reminder of this paper is structured as follows: After this introduction, Section **??** introduces a set of basic definitions we use along the paper. Section **??** presents a motivating scenario that illustrates both the problem and the solution tacked in this paper. Section **??** introduces the foundations of our approach.

Section X validates the approach on DSLs we take from GitHub. Section X discusses the threads to validity. Section X presents the related work and, finally, Section X 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. M. V. Cengarle, H. Grönniger, and B. Rumpe. Variability within modeling language definitions. In A. Sch¸rr and B. Selic, editors, *Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, pages 670–684. Springer Berlin Heidelberg, 2009.
4. B. Combemale, J. Deantoni, B. Baudry, R. France, J.-M. Jézéquel, and J. Gray. Globalizing modeling languages. *Computer*, 47(6):68–71, June 2014.
5. 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.
6. 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.
7. 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.
8. L. Hamann, O. Hofrichter, and M. Gogolla. On integrating structure and behavior modeling with ocl. In R. France, J. Kazmeier, R. Breu, and C. Atkinson, editors, *Model Driven Engineering Languages and Systems*, volume 7590 of *Lecture Notes in Computer Science*, pages 235–251. Springer Berlin Heidelberg, 2012.
9. 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.
10. 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.
11. J. Liebig, R. Daniel, and S. Apel. Feature-oriented language families: A case study. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 11:1–11:8, New York, NY, USA, 2013. ACM.

12. 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.
13. 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.
14. K.-D. Schewe and J. Zhao. Typed abstract state machines for data-intensive applications. *Knowledge and Information Systems*, 15(3):381–391, 2008.
15. E. Vacchi, W. Cazzola, B. Combemale, and M. Acher. Automating Variability Model Inference for Component-Based Language Implementations. In P. Heymans and J. Rubin, editors, *SPLC'14 - 18th International Software Product Line Conference*, Florence, Italie, Sept. 2014. ACM.
16. E. Vacchi, W. Cazzola, S. Pillay, and B. Combemale. Variability support in domain-specific language development. In M. Erwig, R. Paige, and E. Wyk, editors, *Software Language Engineering*, volume 8225 of *Lecture Notes in Computer Science*, pages 76–95. Springer International Publishing, 2013.
17. 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.
18. J. White, J. H. Hill, J. Gray, S. Tambe, A. Gokhale, and D. Schmidt. Improving domain-specific language reuse with software product line techniques. *Software, IEEE*, 26(4):47–53, July 2009.
19. 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.