Puzzle: A tool for analyzing and extracting specification clones in DSLs

David Méndez-Acuña, José Angel Galindo Duarte, Benoit Combemale, Arnaud Blouin, Benoit Baudry

Puzzle: A tool for analyzing and extracting specification clones in DSLs

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

INRIA and University of Rennes 1. France

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

Abstract. The use of domain-specific languages (DSLs) is a successful technique in the development of complex systems. Indeed, the construction of new DSLs addressing the particular needs of software projects has become a recurrent activity. In this context, the phenomenon of *specification cloning* has started to appear. Language designers often copy&paste some parts of the specification from legacy DSLs to "reuse" formerly defined language constructs. As well known, this type of practices introduce problems such as bugs propagation, thus increasing of maintenance costs. In this paper, we present Puzzle, a tool that uses static analysis to facilitate the detection of specification clones in DSLs implemented under the executable metamodeling paradigm. Puzzle also enables the extraction specification clones as reusable language modules that can be later used to build up new DSLs.

1 Introduction

A domain-specific language (DSL) is a software language whose expressiveness is limited to a well-defined domain. A DSL offers the abstractions (a.k.a., *language constructs*) needed to describe an aspect of a system under construction. The use of DSLs has become a successful technique to achieve separation of concerns in the development of complex systems [5].

Naturally, the adoption of such a language-oriented vision relies on the availability of the DSLs necessary to describe all the aspects of the system under construction [3]. As a result, the DSLs development has become a frequent activity in software projects [7]. In this context, the phenomenon of *specification cloning* has started to appear. Language designers often copy&paste some parts of the specification from legacy DSLs with the objective to "reuse" formerly defined language constructs. This practice might have some problems such as bug replications that increase maintenance costs [11].

Ideally, reuse should correspond to a systematic practice where the language constructs that are used in more than one DSL are defined in interdependent language modules that can be used as plug-in pieces during the DSLs development process. In this paper, we present Puzzle, a tool to assist refactoring processes intended to remove specification clones in a given set of legacy DSLs. More precisely, Puzzle offers the following features:

Detection of specification clones. Puzzle provides a set of comparison operators that enable automatic detection of specification clones in a given set of DSLs. These operators take into account not only the names of the constructs, but also the inter-constructs relationships and their semantics. Additionally, the implementation of Puzzle is flexible enough to permit the definition of new comparison operators. Hence, the detection strategy can be easily improved or adapted to particular contexts.

Quantification of potential reuse. Puzzle comes out with a set of metrics (inspired in [1]) to quantify the potential reuse emerging from the existing specification clones. The objective is to provide a mechanism that allows language designers to estimate (in an objective fashion) the benefit of a refactoring process intended to remove specification clones in a given set of DSLs. For example, Puzzle measures the amount and percentage of language constructs cloned in a set of DSLs, as well as how different is a given DSL with respect to the others. All these metrics are presented in the form of charts implemented as HTML reports that can be easily shared and published.

Extraction of reusable language modules. Puzzle enables a reverse engineering process to extract reusable language modules from the detected specification clones [8]. This strategy is based on a principle illustrated in Figure 1: if a DSL specification is viewed as a set of specification elements, then specification clones can be viewed as sets overlapping, and reusable language modules can be obtained by breaking down that overlapping [10]. The language modules resulting from this refactoring process can be later assembled in the construction of new DSLs.

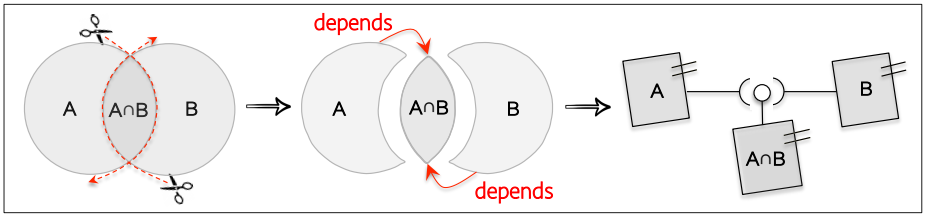

Fig. 1: Breaking down overlapping for obtaining reusable language modules

2 Puzzle

Technological space. Like general purpose languages, DSLs are implemented in terms of syntax and semantics. Nowadays, there are diverse technological spaces available for the implementation of such implementation concerns [9]. Puzzle supports DSLs such that the syntax is specified through metamodels whereas semantics is specified operationally through *domain-specific actions* [4].

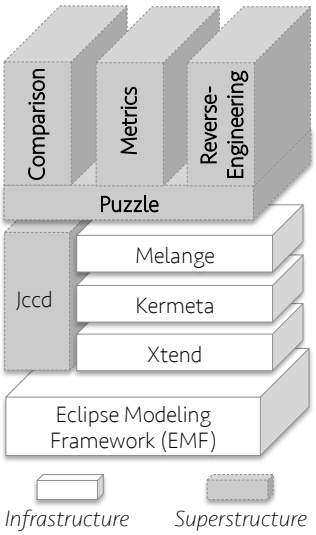

Fig. 2: Tool's architecture

Architecture. The architecture of PUZZLE is composed of two parts illustrated in Figure 2: the infrastructure and the superstructure. The *infrastructure* is a set of plug-ins that enable the specification of DSLs according to the technological space described above. In turn, the *superstructure* is a set of plug-ins that provides analysis and reverse-engineering techniques on the DSLs specified on top of the infrastructure.

The PUZZLE's infrastructure is based on the Eclipse Modeling Framework (EMF). EMF provides a modeling language, called Ecore, which we use to specify metamodels. In turn, we use the notion of aspects provided Kermeta [6] to specify operational semantics. An aspect encapsulates a set of domain-specific actions that are weaved into a metaclass of the metamodel. The mapping between metamodels and aspects is specified in Melange[1].

In turn, the superstructure of PUZZLE corresponds to a set plug-ins that can be divided into three categories according to their functionalities: comparison, metrics, and reverse-engineering. Comparison plug-ins implement the comparison operators needed to detect specification clones at the level of abstract syntax and semantics (for the case of comparison of semantics, Puzzle uses JCCD [2]. The metrics plug-ins compute a set of metrics for the detected specification clones and present the results as a set of HTML reports that display those metrics in the form of charts. The reverse-engineering plug-ins implement the algorithms that extract reusable language modules from the detected specification clones.

Tool demonstration. In the rest of this section, we provide three videos (available in the papers' website[2]) that show the way in which a set of DSL defined in the PUZZLE's infrastructure is analyzed by the PUZZLE's superstructure. The PUZZLE's source code is available in the project's website[3].

The input of PUZZLE is a Melange script that references a set of DSLs. The analysis starts by comparing the DSLs specifications (at the level of the abstract syntax and the semantics) and produces a first report indicating whether there are any specification clones or not. This report looks like a Venn diagram where each DSL is represented by a set, and intersections among sets indicate specification clones (*video 1: detecting specification clones*). Then, a set of metrics is computed from those specification clones. These metrics are intended to quantify the specification clones among the DSLs to objectively measure the associated potential reuse (*video 2: measuring specification clones*). Finally, a set of reusable

[1] Melange website: http://melange-lang.org/
[2] **Tool demonstration:** http://puzzle-demo.weebly.com/
[3] Puzzle's website: http://damende.github.io/puzzle/

language modules is extracted from those specification clones. Those language modules can be later assembled among them to produce new DSLs (*video 3: Reverse-engineering reusable language modules*).

Acknowledgments

This work is supported by the ANR INS Project GEMOC (ANR-12-INSE-0011), the bilateral collaboration VaryMDE between Inria and Thales, and the bilateral collaboration FPML between Inria and DGA.

References

1. C. Berger, H. Rendel, B. Rumpe, C. Busse, T. Jablonski, and F. Wolf. Product line metrics for legacy software in practice. In *Workshop Proceedings of the International Software Product Lines Conference*, SPLC 2010, pages 247–250, Jeju Island, South Korea, 2010. Springer.
2. B. Biegel and S. Diehl. Jccd: A flexible and extensible api for implementing custom code clone detectors. In *Proceedings of the International Conference on Automated Software Engineering*, ASE 2010, pages 167–168, Antwerp, Belgium, 2010. ACM.
3. T. Clark and B. S. Barn. *Domain Engineering: Product Lines, Languages, and Conceptual Models*, chapter Domain Engineering for Software Tools, pages 187–209. Springer, Berlin, Heidelberg, 2013.
4. B. Combemale, C. Hardebolle, C. Jacquet, F. Boulanger, and B. Baudry. Bridging the chasm between executable metamodeling and models of computation. In *Proceedings of the International Conference on Software Language Engineering*, SLE 2012, pages 184–203, Dresden, Germany, 2013. Springer.
5. S. Cook. Separating concerns with domain specific languages. In *Proceedings of the Joint Modular Languages Conference*, JMLC 2006, pages 1–3, Oxford, UK, 2006. Springer.
6. J.-M. Jézéquel, B. Combemale, O. Barais, M. Monperrus, and F. Fouquet. Mashup of metalanguages and its implementation in the kermeta language workbench. *Software & Systems Modeling*, 14(2):905–920, 2015.
7. J.-M. Jézéquel, D. Méndez-Acuña, T. Degueule, B. Combemale, and O. Barais. When systems engineering meets software language engineering. In *Proceedings of the International Conference on Complex Systems Design & Management*, CSD&M 2014, pages 1–13, Paris, France, 2015. Springer.
8. D. Méndez-Acuña, J. A. Galindo, B. Combemale, A. Blouin, and B. Baudry. Reverse-engineering reusable language modules from legacy domain-specific languages. In *Proceedings of the International Conference on Software Reuse*, ICSR 2016. Springer, Limassol, Cyprus, 2016.
9. M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4), 2005.
10. 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.
11. S. Zschaler, D. S. Kolovos, N. Drivalos, R. F. Paige, and A. Rashid. Domain-specific metamodelling languages for software language engineering. In *Proceedings of the International Conference on Software Language Engineering*, SLE 2009, pages 334–353, Denver, CO, USA, 2010. Springer.