A Longitudinal Field Study on Creation and Use of Domain-Specific Languages in Industry

Jasper Denkers
Delft University of Technology & Océ Technologies B.V.
Delft, The Netherlands
j.denkers@tudelft.nl

ABSTRACT

Domain-specific languages (DSLs) have extensively been investigated in research and have frequently been applied in practice for over 20 years. While DSLs have been attributed improvements in terms of productivity, maintainability, and taming accidental complexity, surprisingly, we know little about their actual impact on the software engineering practice. This PhD project, that is done in close collaboration with our industrial partner Océ - A Canon Company, offers a unique opportunity to study the application of DSLs using a longitudinal field study. In particular, we focus on introducing DSLs with language workbenches, i.e., infrastructures for designing and deploying DSLs, for projects that are already running for several years and for which extensive domain analysis outcomes are available. In doing so, we expect to gain a novel perspective on DSLs in practice. Additionally, we aim to derive best practices for DSL development and to identify and overcome limitations in the current state-of-the-art tooling for DSLs.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**.

KEYWORDS

domain-specific languages, language workbenches, model-driven engineering, longitudinal field study

ACM Reference Format:
Jasper Denkers. 2019. A Longitudinal Field Study on Creation and Use of Domain-Specific Languages in Industry. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3338906.3341463

1 INTRODUCTION

Developing software in complex domains becomes harder when the scale at which it is being performed increases. At the same time, the value of many products is highly reliant on the quality and capabilities of the software that ships with it. In light of the ever increasing demand from society for more advanced products, these factors indicate the importance of advancing the most important tools software engineers use: programming languages.

Encoding business domain concepts into these programming language is a core challenge in software engineering [15]. Model-driven engineering (MDE) [8, 11] is a method in which such concepts are identified and encoded on a higher level of abstraction than what typically would occur in a general purpose programming language (GPL). It is based on introducing programming constructs for specific domains, enabling the modeling of business concepts more effectively. Such constructs enable representing crucial parts of complex systems better in software, aimed at improving the overall software engineering process in terms of quality, productivity, and maintainability. A particular direction of applying MDE is using domain-specific languages (DSLs) [5]. DSLs embed the encoding of business abstractions in specific tailor made programming languages. However, developing and applying DSLs in practice is non-trivial due to the requirement of language engineering skills. Language workbenches are tools that promise to make language engineering more accessible by providing infrastructures for developing and deploying DSLs. While using language workbenches has a big potential, clear methodology for their application in practice is lacking.

We propose to evaluate the state-of-the-art in language workbenches for developing DSLs. The context of the research is our industrial partner Océ, a large digital printer manufacturing company. Océ's software research and development department has experience in applying MDE for over 10 years, and in some projects also with using DSLs. More recently, the company experiments with developing such DSLs with language workbenches. While early results are promising, the company struggles to scale the adoption of the technology. Training engineers is hard. The engineers that develop DSLs feel like having to "reinvent the wheel" for particular language engineering tasks like code generation, because best practices and clear DSL design patterns are lacking. It is unclear to what extent tool characteristics influence the success of introducing DSLs. Despite the widely recognised potential of DSLs and language workbenches, the above problems still make investing in DSL technology in industry risky.

By applying MDE in many projects, Océ obtained extensive domain knowledge for complex domains like modeling behavior, performance, and physical aspects of printing systems. The domain analysis outcomes of these projects are valuable assets for experimenting with DSL development. In addition, the environment has the potential for big impact by DSL solutions. For example, printers interface with finishing devices (e.g. booklet makers) that are produced by external companies. The integration between such

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3341463

finishing devices and Océ printers has a significant software component and it is currently a time and cost expensive process. Therefore, there are also interesting opportunities to experiment with the usage of DSLs as a means to integrate externally produced devices with the software from Océ, potentially with the external companies as users of such DSLs.

The environment at Océ thus provides a unique opportunity to study DSL development with language workbenches. Our objective is to identify best practices and development patterns. In particular, we qualitatively compare their impact on the software engineering process with respect to more traditional MDE approaches. In addition, we aim to identify limitations in current state-of-the-art tooling. Our approach is a longitudal field study [10] in which we perform multiple DSL engineering case studies at Océ. For these cases, we focus on a particular type of projects. These projects should apply MDE techniques, for which domain analysis has already been carried out, and for which an existing implementation using traditional technologies is already present. Second, the projects do not already use a DSL, or if they do use a DSL, it is not developed using a language workbench. Finally, the projects involve a domain that is sufficiently complex such that the engineers working on it struggle with advancing their solutions in the existing approach.

2 BACKGROUND

Domain-Specific Languages (DSLs) [5]. DSLs are programming languages specific for a particular (business) domain. Higher levels of abstractions in these languages, in contrast to general purpose languages (GPLs), provide opportunity for improving software engineering on several aspects, e.g., productivity and quality. Additionally, DSLs can help to apply development methodologies that depend on abstractions, e.g., MDE. For adopting DSLs, the engineers applying them ideally have full-fledged integrated development environments (IDEs) including editor services like syntax highlighting, interactive error reporting, and refactorings.

Language workbenches (LWBs) [4]. LWBs are tools specific for developing DSLs and deriving IDEs. They provide convenient methods to implement syntax, semantics, transformations, and code generation for languages. LWBs that support language modularity and composition, enabling reuse language components, lower the threshold to start developing DSLs with good software engineering practices. Language workbenches differ e.g. in supported notations (textual, graphical, or projectional) and maintainers (academic or industrial). An overview and comparison of language workbenches is given in [2, 3].

3 RESEARCH QUESTIONS

The underlying goal of this work is to advance the capabilities of handling software engineering complexity. To this extent, we investigate a particular approach to achieve this goal: apply domain-specific languages developed with language workbenches on MDE projects at Océ. This leads to several software engineering research questions related to *usefulness* and *feasibility*, which we categorize by the types as identified by Shaw [13].

Our first research question is of type "methods or means of development":

> **RQ-Patterns:** *What are useful patterns for developing DSLs with language workbenches for MDE in an industrial setting?*

While many well-known and evaluated design patterns are available for general software engineering [6], the available patterns specific to DSL engineering are limited. This hinders the development of DSLs, since engineers new to the technology feel like having to "reinvent the wheel". In our work we aim to develop and evaluate patterns in DSL development that have the potential of being generally *useful*.

Our other three research questions are of the type "design, evaluation, or analysis of a particular instance":

> **RQ-Effectiveness:** *Do DSLs developed with language workbenches improve MDE in an industrial setting in terms of software engineering productivity, quality, and maintainabilty?*

For maintaining a competitive market position, the quality and capabilities of the products Océ produces is most important. Therefore, we focus on the effectiveness of DSL solutions to improve the company's software engineering to deliver advanced products. Using DSLs is only *useful* if these capabilities actually improve. In particular, we focus on comparing effectiveness in comparison with existing MDE approaches present at Océ.

> **RQ-ROI:** *Does the benefit gain of DSLs using language workbenches in an industrial setting outweighs their development cost? Is there return on investment (ROI)?*

While applying DSLs might be useful, they have a certain development cost. It is only *feasible* in practice to apply DSL technology if the value improvement or effort reduction they introduce outweigh the cost required to develop the DSLs.

> **RQ-Tooling:** *Are state-of-the-art language workbenches useful for developing DSLs for MDE in an industrial setting? What are their limitations?*

We focus on language workbenches as the tools to develop DSLs and to derive IDEs for the DSLs. They intend to decrease the effort required to develop DSLs and to improve the quality, maintainability, and usability of DSL implementations. With this question we focus on evaluating the state-of-the-art in language workbenches.

4 METHODOLOGY

We apply the *design research* method. In particular, we instantiate Hevner's design science framework [21] (Figure 1). It incorporates behavioral aspects, relevant for doing research in an industrial environment. The proposed research builds on existing foundations and methodologies for developing DSLs. In addition, it takes the business needs that arise from the environment into account. Given both sources, we will perform several iterations of case studies of developing and evaluating DSL artifacts. The artifacts will be applied at Océ and thus have to contribute to the industrial environment. Answers to the research questions will contribute back to the knowledge base.

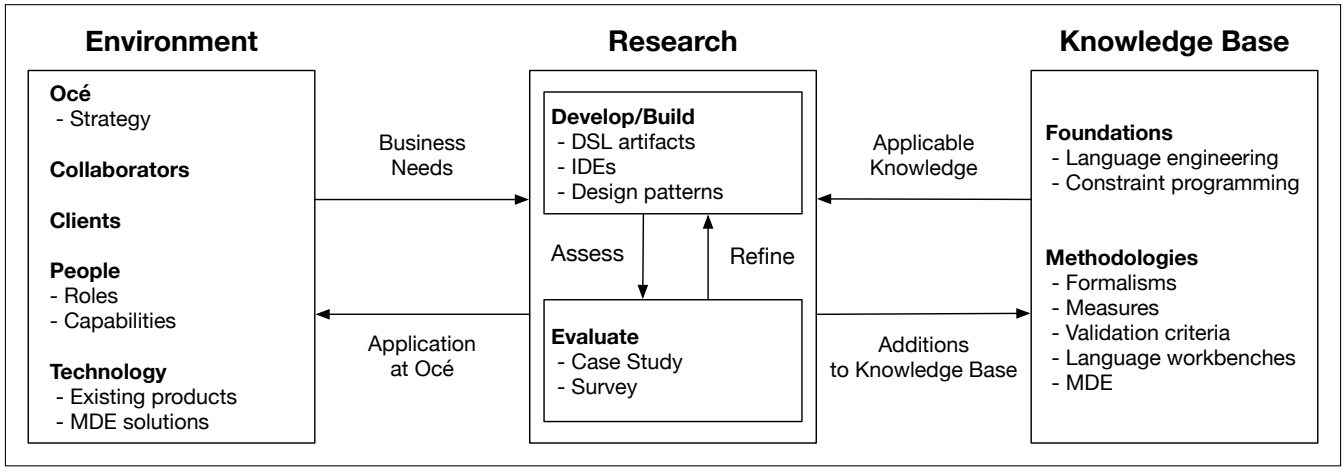

Figure 1: An instantiation of Hevner et al.'s design science framework [21] for the proposed research.

Environment. Océ is characterized by the development of high-end products. This is part of the company's strategy and implies the need for development of advanced software. There are multiple categories of potential DSL users: Océ's own employees, external companies producing devices that interface with Océ printers, and clients that configure their systems. Last, an important part of the environment is the software for existing products and the MDE solutions that are used. These aspects define the problem space in which we perform our research.

Knowledge Base. We build on a knowledge base consisting of foundations and methodologies relevant to our work. In particular, we apply existing language engineering techniques and tools. In addition, we experiment with combinations of technologies. For example, a DSL with constraint solving as a backend enables declarative notations, and fuses DSL technology with constraint programming.

DSL Implementations. We use the Spoofax language workbench [7, 16] for the development of our DSLs. This limits the scope of the tooling perspective and allows us to focus on evaluation of a particular type of language workbench. For Spoofax, important characteristics are that notation is textual and the maintaining party is a research group.

5 RELATED WORK

Voelter et al. reported on several industrial applications of the MPS language workbench. mbeddr [20] introduces extensions to the C language that introduce abstractions specific for the domain of embedded software. The authors evaluate mbeddr in the context of developing a smart electricity meter [17]. They found that it helps significantly in managing complexity, it improves testability, it reduces development effort, and all without introducing memory or performance overhead. Besides the positive outcomes, the authors note that adopting mbeddr might be difficult due to developers lacking skills and because of adaptions to the development process. The authors also evaluate the language workbench itself for developing mbeddr rather than evaluate the case on the language level [19]. The outcomes are generally positive: language modularity helps managing complexity, the advantages of projectional editing outweigh its drawbacks, and using declarative meta-DSLs for implementing certain DSLs aspects help in managing language implementation complexity. Most recent, the authors propose an architecture for the application of MPS for implementing a DSL in a safety-critical domain [18]. An industrial case study in the healthcare domain validates the architecture and the authors plan to apply the approach in other domains as well.

Additionally, Kelly and Tolvanen reported on several industrial case studies performed with the MetaEdit+ language workbench. In [14], they focus on evaluating effort. For ten cases, the required implementation effort was between 3 and 15 man-days, with an average of 10 days.

Schuts' thesis [12] reports on the experiences of applying DSLs in industry for system evolution. DSLs have been developed using the Xtext language workbench and they found that DSLs provide a way to solve industrial problems in less time that with previous ways of working. For several projects the outcome had a positive ROI.

The aforementioned recent work evaluated the impact of applying DSLs using LWBs in industrial software engineering. Their focus was on specific language workbenches with distinct notation formats. First, MPS, based on projectional editing. Second, MetaEdit+, based on graphical editing. Third, Xtext, based on textual notations. Amongst notation, the language workbenches differ on other aspects. While the conclusions drawn in these studies are generally positive, they provide too little evidence to generalize over the findings [9]. It is unclear to what extent the results are biased by tool characteristics. All authors mentioned in this section indicate the need for additional studies with applications in other contexts and with applying other tools.

6 CURRENT ACHIEVEMENTS

The author of this work is in his second year of his PhD. In the first year, he worked on the migration of existing DSL implementations

developed and used at Océ to a language workbench [1]. In particular, this case study involved languages for defining interfaces (IDL) and modeling system behavior (OIL) that were implemented with XML for syntax and Python for static analysis and code generation. The target language workbench was Spoofax. Besides migrating the implementations, he introduced a new concise DSL syntax and additional functionality, including cross-language static analysis. This work contributed a general pattern for adopting language workbenches for already existing DSLs implemented with custom technologies (**RQ-Patterns**). The author is currently involved in the further development of OIL. At the time of writing, a paper is in progress that presents a general pattern for implementing model transformation pipelines (**RQ-Patterns**). The language is nearing the stage at which it is ready for application in production. From that point, the evaluation considering the other research questions will start.

In parallel, the author is working on a project called Sheela (*Shee*t *La*nguage). This project focuses on the modeling of configurations of printers and finishers and concerns their integration. The existing MDE solutions are several versions of C# frameworks that model physical aspects of sheets and actions on sheets required for integrating printers with finishers. In this project, we experiment with a declarative DSL with constraint solving as the backend (**RQ-Patterns**). The aim is to have executable specifications, automatic validation, and deriving optimal parameters for jobs requested for given specifications (**RQ-Effectiveness**). Currently, Océ software engineers integrate finishers using the C# framework, which is a costly process. With the new DSL, the objective is to improve this process, and investigate the possibility to have external finisher producing companies as users of the DSL to reduce overall integration costs (**RQ-ROI**).

7 EXPECTED CONTRIBUTIONS

Our intended research contributions come from several DSL artifacts. First, we intend to contribute design patterns that are evaluated in several case studies. Second, we aim to report on the evaluation of the state-of-the-art in language workbenches in an industrial setting. Third, we expect to contribute to tooling with respect to the ability of handling large scale models. Finally, the author intends to summarize his findings in a *technology transfer* paper in the final year of his PhD. This will report from a higher perspective on the performed case studies and the observations made. The goal is that it will provide useful insights in what impacts adopting LWBs for DSLs in industry by giving an overview of the lessons learned from the case studies performed at Océ.

8 EVALUATION

We evaluate **RQ-Patterns** by assessing the usefulness of the application of the introduced patterns in the case study DSLs. We evaluate **RQ-Effectiveness** by qualitatively comparing DSL solutions with their existing MDE engineering counterparts. In addition, where possible, we survey DSL users and engineers about their experiences with both old and new approaches. For **RQ-ROI**, we measure the effort for developing DSLs. We estimate the value improvement or effort reduction that is the result of the DSLs to get an indication of return on investment. We evaluate tooling (**RQ-Tooling**) by benchmarking implementations on large models and by surveying users of the language workbenches on both the meta level (language engineers) and user level (language users).

ACKNOWLEDGEMENTS

The author is advised by Prof. Dr. Eelco Visser and Dr. Andy Zaidman. This research is funded by a grant from the Top Consortia for Knowledge and Innovation (TKIs) of the Dutch Ministry of Economic Affairs and from Océ.

REFERENCES

[1] Jasper Denkers, Louis van Gool, and Eelco Visser. 2018. Migrating custom DSL implementations to a language workbench (tool demo). In *Proc. ACM SIGPLAN International Conference on Software Language Engineering*. ACM, 205–209.
[2] Sebastian Erdweg, Tijs Van Der Storm, Markus Völter, Meinte Boersma, Remi Bosman, William R Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, et al. 2013. The state of the art in language workbenches. In *International Conference on Software Language Engineering*. Springer, 197–217.
[3] Sebastian Erdweg, Tijs Van Der Storm, Markus Völter, Laurence Tratt, Remi Bosman, William R Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, et al. 2015. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. *Computer Languages, Systems & Structures* 44 (2015), 24–47.
[4] Martin Fowler. 2005. Language workbenches: The killer-app for domain specific languages. (2005).
[5] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.
[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software Addison-Wesley. *Reading, MA* (1995), 1995.
[7] Lennart CL Kats and Eelco Visser. 2010. The spoofax language workbench: rules for declarative specification of languages and IDEs. In *ACM sigplan notices*, Vol. 45. ACM, 444–463.
[8] Stuart Kent. 2002. Model driven engineering. In *International Conference on Integrated Formal Methods*. Springer, 286–298.
[9] Parastoo Mohagheghi and Vegard Dehlen. 2008. Where is the proof?-A review of experiences from applying MDE in industry. In *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 432–443.
[10] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. 2012. Case study research in software engineering. In *Guidelines and examples*. Wiley Online Library.
[11] Douglas C Schmidt. 2006. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY*- 39, 2 (2006), 25.
[12] MTW Schuts. 2017. *Industrial Experiences in Applying Domain Specific Languages for System Evolution*. Ph.D. Dissertation. [Sl: sn].
[13] Mary Shaw. 2003. Writing good software engineering research papers. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 726–736.
[14] Juha-Pekka Tolvanen and Steven Kelly. 2018. Effort Used to Create Domain-Specific Modeling Languages. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 235–244.
[15] Eelco Visser. 2015. Understanding software through linguistic abstraction. *Science of Computer Programming* 97 (2015), 11–16.
[16] Eelco Visser, Guido Wachsmuth, Andrew Tolmach, Pierre Neron, Vlad Vergu, Augusto Passalaqua, and Gabriël Konat. 2014. A Language Designer's Workbench: A One-Stop-Shop for Implementation and Verification of Language Designs. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. ACM, 95–111.
[17] Markus Voelter, Arie van Deursen, Bernd Kolb, and Stephan Eberle. 2015. Using C language extensions for developing embedded software: a case study. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 655–674.
[18] Markus Voelter, Bernd Kolb, Klaus Birken, Federico Tomassetti, Patrick Alff, Laurent Wiart, Andreas Wortmann, and Arne Nordmann. 2018. Using language workbenches and domain-specific languages for safety-critical software development. *Software & Systems Modeling* (2018), 1–24.
[19] Markus Voelter, Bernd Kolb, Tamás Szabó, Daniel Ratiu, and Arie van Deursen. 2017. Lessons learned from developing mbeddr: a case study in language engineering with MPS. *Software & Systems Modeling* (2017), 1–46.
[20] Markus Voelter, Daniel Ratiu, Bernd Kolb, and Bernhard Schaetz. 2013. mbeddr: Instantiating a language workbench in the embedded software domain. *Automated Software Engineering* 20, 3 (2013), 339–390.
[21] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. 2004. Design science in information systems research. *MIS quarterly* 28, 1 (2004), 75–105.