An Integrated Tool Environment for Experimentation in Domain Specific Language Engineering

Florian Häser, Michael Felderer, Ruth Breu
Institute of Computer Science
University of Innsbruck
{florian.haeser, michael.felderer, ruth.breu}@uibk.ac.at

ABSTRACT

Domain specific languages (DSLs) are widely used in practice and investigated in software engineering research. But so far, language workbenches do not provide sufficient built-in decision support for language design and improvement. Controlled experiments have the potential to provide appropriate, data-driven decision support for language engineers and researchers to compare different language features with evidence-based feedback. This paper provides an integrated end-to-end tool environment to perform controlled experiments in DSL engineering. The experiment environment is built on the basis and integrated into the language workbench Meta Programming System (MPS). The environment not only supports language design but also all steps of experimentation, i.e., planning, operation, analysis & interpretation, as well as presentation & package. The tool environment is presented by means of a running example experiment comparing the time taken to create system acceptance tests for web applications in two different DSLs.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging.

Keywords

Domain Specific Languages (DSLs), Language Engineering, Experimentation, Tool Support, Meta Programming System (MPS), Empirical Evaluation, Controlled Experiment

1. INTRODUCTION

Domain specific languages (DSLs) allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Since the advent of language workbenches, DSLs are not only widely used in practice [18], but also a hot topic in research [3]. Industry-ready language workbenches like the Meta Programming System (MPS) [10] support a language engineer in the definition of the syntax and semantics of a (textual) domain specific language, and in the creation of DSL-specific editors with remarkable benefit for the language user defining constructs in the DSL.

But so far, language workbenches do not provide sufficient decision support for language design and adaptation, which is essential for successful language engineering. For a language engineer, it is not always easy to determine in advance, what effects, e.g., on creation time or the resulting document length of DSL scripts, design decisions or changes to a DSL have. Furthermore, according to recent systematic literature reviews [7, 11], evaluation of usability is often neglected for DSLs.

Controlled experiments [19] have the potential to support language engineers and researchers to compare different DSL features and to support decisions on language design and adaptation by data-driven and evidence-based feedback. Over the last years, the number of experiments performed in software engineering increased, which significantly extended the body of knowledge on benefits and limitations of software engineering methods and techniques [19]. For the field of DSL engineering, this is however, not entirely true, where experiments are still rare [11]. We think that one main reason for this is missing integrated tool support which enables researchers but especially also language engineers in practice to conduct experiments directly in a DSL workbench in a straightforward, but methodologically sound way. As a positive side effect, such an integrated end-to-end tool environment also facilitates replication of experiments as the experimental setting can automatically be interchanged and reproduced.

According to a systematic literature review on automated support for controlled experiments conducted by Freire et al. [6] there are already methodologies and tools available, which support parts or even the entire experimentation process. However, those approaches are limited regarding to experiments with DSLs. DSL engineering is not performed in the web and its empirical analysis also includes in addition to questionnaires, automatic measurements, as only produced in a language workbench.

This paper contributes by providing an integrated end-to-end tool environment to perform controlled experiments in domain specific language engineering inside the DSL workbench MPS. We integrate existing established DSLs for experimentation to provide additional value by providing for the first time an environment which uses the same tooling, i.e., MPS, to create or adapt DSLs for specific purposes and to perform experiments on them to support language engineering decisions.

Based on the mixed notations of MPS supporting textual, tabular and graphical representation of concepts, our environment supports not only DSL design (providing the experimental object in our case), but all phases of experimentation, i.e., planning, operation, analysis & interpretation, and presentation & package. Our environment is based on established DSLs and tools which are all integrated into MPS. For planning experiments, the established DSL ExpDSL [5] is applied to formulate goals and hypotheses. In addition, for

EASE '16, June 01-03, 2016, Limerick, Ireland

DOI: http://dx.doi.org/10.1145/2915970.2916010

statistical analysis metaR [1] is applied to simplify the use of complex R functions for non-experts like most language engineers. For the experiment operation, MPS is extended to monitor metrics like the time taken to create documents or their length. For analysis & interpretation, R output is integrated into MPS, and finally for presentation & package a DSL for LaTeX is applied [17]. The integration of several DSLs and representation is enabled by the projectional editors of MPS, which allow the integration of languages, but also alternative representations of data, such as plots, all within one single document (see Figure 4). The full MPS integration enables practitioners and researcher to conduct experiments in an easy way without media breaks and with different levels of rigor.

The remainder of this paper is organized as follows. Section 2 gives an overview on related approaches. In Section 3 we introduce an example case for DSL experimentation in the domain of acceptance testing of web applications, which we use as a running example throughout this paper. Section 4 presents the architecture, procedure and the integrated tool itself. For better understandability the case from Section 3 is applied to the steps on conducting a controlled experiment. Finally, Section 5 concludes the paper and presents future work and visions for the tool environment.

2. RELATED WORK

Controlled experiments are increasingly used to gain empirical evidence on software engineering phenomena. A recent systematic literature review conducted by Freire et al. [6] reveals however that tool support for planning and conducting experiments is very limited. This holds especially also for DSL engineering, where conducting experiments is not only relevant for researchers but especially also for language engineers as pointed out in the previous section. Most tools for experimentation are dedicated to the planning phase, by supporting the experimenter with some sort of language or ontology. That support is either graphical as proposed by Cartaxo et al. [2] in their empirical software engineering modeling language or textual as of the approaches of Garcia et al. [8] and Siy and Wu [14]. Both present ontologies that allow the definition of an experiment and checking predicates or constraints regarding the validity of the experiment.

Based on their previous research Freire et al. [4] overcame the downsides that none of the previous approaches, provided an integrated environment for all required steps of an experiment, i.e., planning, execution and interpretation by developing their ExpDSL approach. Additionally, in contrast to the other mentioned languages and ontologies, completeness and expressibility was empirically evaluated [5].

On the contrary to above mentioned approaches, regarding the planning of experiments and enabling its reproducibility, efforts in developing experiment environments for different applications and domains have been taken.

Hochstein et al. [9] developed a set of integrated tools to support experiments in software engineering for the domain of high-performance computing. Although the proposed Experiment Manager Framework supports quite all of the activities that are carried out in a controlled experiment it has some limitations. Adaption to a different domain such as domain specific language engineering is quite difficult, as, for instance, automated data capturing is custom-built for source code for high performance computing. Another limitation is the lack of support for extending the statistical analysis for other statistical design types than the built-ins.

A web based experiment environment, the SESE (Simula Experiment Support Environment), was proposed by Sjøberg et al. [15]. The environment guides the subject through the assignments of an experiment, measures the time spent for each task, allows to download additional information and the centralized collection of generated artifacts such as source code. The proposed solution allows a broad range of application in very different domains. It is however not perfectly suited for experiments with DSLs, as those experiments not only rely on time measuring for executing a certain task. Valuable metrics, related to the use of a certain language, may include the average thinking time on certain keyword. For automatically capturing such metrics a web based tool is not suitable, actually it has to reside in the actual language workbench.

Because of the limitation of existing approaches, regarding experimentation with DSLs this paper provides an experiment platform decided to DSL experimentation. ExpDSL as of Freire et al. [4], i.e., the DSL for planning and describing formally the actual experiment forms the template for our tool chain. It has been already empirically evaluated and successfully used in a number of experiments. However, ExpDSL only uses DSLs for describing experiments but is not specifically dedicated to DSL experimentation, i.e., the evaluation of concrete DSL language decisions itself. Also from a technological point of view ExpDSL differs from our approach as it is based on XText, whereas our environment is based on MPS with its features to integrate different languages and representations.

3. EXAMPLE EXPERIMENT DESCRIPTION

We consider the adaptation of a domain specific language for acceptance testing of web applications as a concrete practical scenario for DSL experimentation, which was actually performed in our integrated tool environment and presented in this paper. An important requirement for the adaptation of a test design language is that the time to define a test case in the adapted language is not significantly higher than before. We are aware of the fact that time, measured for the creation of a test case, can be influenced by many factors, e.g. usability issues or understanding problems regarding the assignment. Such a variable should be balanced in real experiments always with other variables such as number of atomic steps or quality in terms of readability, understandability and correctness. Nevertheless, also because of the space limitation, the analysis of time is a well suited variable in a simple running example, an excerpt from a real conducted student experiment, for presenting the work-flow and features of the tool environment. In the experiment, two similar testing languages called DSL_1 and DSL_2 were implemented in our environment and compared with regard to the time needed to write test cases.

Listing 1 shows a part of a test case written with the simple DSL (DSL_1). It only consists of two predefined keywords, the step keyword, denoted with "-", and the check keyword. Resulting documents can be used to preserve knowledge, in form of natural language instructions, for a future human tester.

Listing 1: Example for the simplified test language

```
Test Case Specification: Participate to student
                         seminar
- open a browser
- check whether it has started successfully
- visit http://www.uibk.ac.at
- check if the site has been loaded
- click on the Link named OLAT
...
```

Figure 2 shows a test case with the extended DSL (DSL_2), in a screenshot taken during the conduction of an experi-

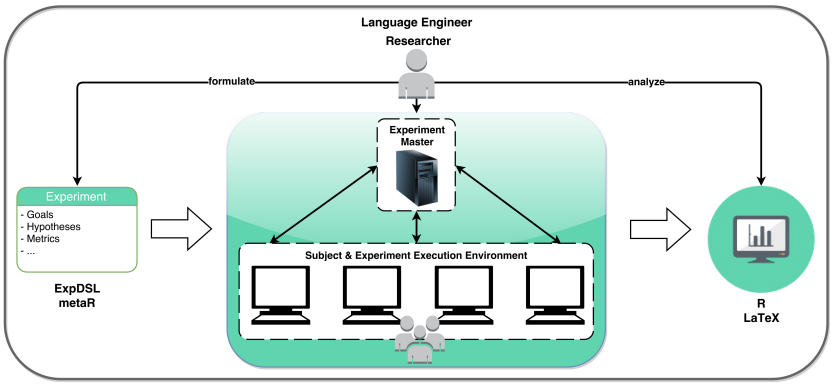

Figure 1: Integrated DSL Experiment Environment Architecture

ment. As one can see on the figures, both languages are suitable for writing acceptance tests for web applications. DSL_1 allows the tester complete freedom in writing the test case and may represent a test language introduced first in a company for providing initial tool support in writing acceptance tests. The second language, i.e., DSL_2, has been enriched with language extensions from the actual application domain that potentially allows a faster definition of tests in that language, compared to DSL_1. It may represent an evolution of the first DSL, developed by a language engineer, for improving tool support in creating acceptance test cases. By intuition one could say that people who use a language tightly bound to the business domain, will certainly work faster than people without that support. Many factors however, such as the introduced restrictiveness or confusion of the user, because concepts are named differently than expected can influence the actual usage.

The language engineer gets data-driven decision support, regarding a possible improvement of the DSL, if an experiment is performed. Tool support such as the experiment platform not only simplifies the conduction of experiments, it also allows easy replication.

Similar experiments could be performed with low effort taking testers or key users also responsible for testing as experimental subjects.

```
test report Export and validate academic record
- open browser
- go to http://www.uibk.ac.at
- fill in username and password
- perform login
- result was success
- select submenu 'Academic Record'
- create academic record for study programme 006461 in german with signature
- check Signature
  - expected 3 green checks
  - actual is 2 green checks 1 red cross
```

Figure 2: Example for the enriched test language

4. INTEGRATED DOMAIN SPECIFIC LANGUAGE ENGINEERING ENVIRONMENT

In this section, we first outline the architecture of the integrated tool environment and then explain the methodology on how to perform a DSL experiment. Each required step is described along with the example case, as presented in the previous section. Both, architecture and methodology are aligned with the steps on planning and executing controlled experiments proposed by Wohlin et al. [19].

Basically, as shown in Figure 1, our integrated environment for controlled experiments with DSLs consists of three major parts for experiment planning, execution and reporting corresponding to the steps of experimentation, where the reporting part performs steps analysis & interpretation as well as presentation & package. As shown in Figure 3, each step is supported by various languages and tools, all accessible from within MPS and implemented as extensions to it.

For experiment planning, we reused the experiment definition language ExpDSL proposed by Freire et al. [5]. ExpDSL was evaluated in 18 controlled experiments and in its second revision the language supports the specification of almost the entire experiment stack. The inclusion of MetaR proposed by Campagne [1] simplifies data analysis with R and makes it accessible to people with limited experience in statistics, as might be the case for many language engineers. For experiment operation, we developed monitoring support to collect metrics which runs and can be configured within MPS. Finally, for reporting, we apply an excerpt of a DSL for LaTeX provided by Völter [17] to automatically create reports and pre-filled LaTeX paper templates including R plots, from which also PDF reports can be generated.

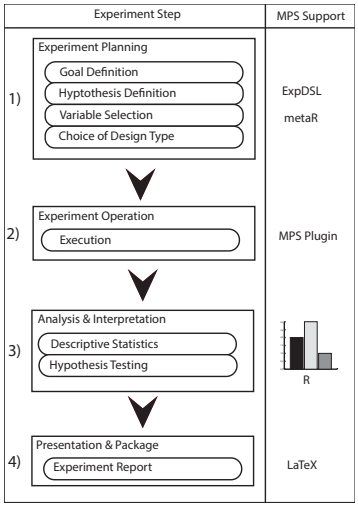

Figure 3: Supported steps

4.1 Methodology

The methodology for experimentation with DSLs supports language engineers and researchers in each step of the con-

trolled experiment procedure as shown in Figure 3. In the following, each step is presented in more detail and exemplified based on the case of changing a DSL for acceptance testing of web applications as described in Section 3.

4.1.1 Goal Definition and Experiment Planning

In order to formulate all relevant aspects of an experiment in a precise way we reused ExpDSL [5]. The language allows a language engineer or researcher to formulate experiment goals, hypotheses, the statistical methods including dependent and independent variables as well as metrics in an intuitive way. Extending the base language with features that are based on metaR [1] and the mapping of specified metrics to concrete data as collected during the operation phase, enables the automated generation of interesting plots and testing of hypotheses.
Some information that is entered during this phase is used for generating a PDF report or a LaTeX template structure pre-filled with information.
Due to space limitations and the fact that a language engineer in practice is often only interested in basic results of an experiment which lack full rigor, but are sufficient to provide decision support, we omit details of statistical analysis in this example. For sure, additional analyses like tests for normality or power analysis can also be performed in our DSL experimentation environment if needed. Our basic example case only shows how to formulate goals and hypotheses (see box below) and how to perform a related hypothesis tests and draw a box plot (see Figure 4).

Abstract This paper presents a controlled experiment...
Goals
G1: **Analyze** the efficiency of similar test DSLs
for the purpose of evaluation
with respect to creation time
from the point of view of a DSL user
in the context of graduate students using assisting editors for test case creation.
Hypotheses
H0: The time to create tests with both DSLs is equal
H1: The time to create tests differs significantly

The DSL highlights relevant keywords, e.g., 'Goals' and 'Hypotheses' and provides a goal template [16]. Based on the independent (i.e., DSL type in our case) and dependent variables (i.e., time taken to create the document in our case), hypotheses are formulated. On this basis the statistical analysis is formulated (see Figure 4). Note that, so far, the prototype only supports the single factor two level design of experiment type. The formulation of the actual design of an experiment takes most of the effort needed for conducting the experiment. During this step, one not only has to specify the statistical methods, but also to declare dependent and independent variables including all related metrics to be collected. These metrics have to be linked to a predefined fixed set of measurable variable. Such measurable variables can be document length, time taken to create or number of deletions within a language artifact. Operating within the language workbench enables to capture many additional measures, for enabling further automatic linking. A feature planned for the next milestone is the automated measuring of the number of deletions and thinking time for each individual language construct.
This code is used to automatically test hypotheses, for generating expressive plots and calculating p-values. Relevant results are then included in the resulting report.

4.1.2 Experiment Operation

The same language workbench the language engineer used to develop his or her language and to plan a related language experiment, is used as execution platform for the experiment during the operation phase. It can operate in either master or client mode (see Figure 1). When starting an experiment, a test subject requests its assignment from the master. During the fulfillment of the assignment, the client collects empirical evidence, such as time measurements or number of deletions within a language fragment. Metrics are collected as defined in Section 4.1.1 during the experiment definition. In order to support the operation phase, particularly for the automated collection of metrics, we provide extensions to the MPS language workbench. Our running example includes a language enabling the linkage of metrics and measurable artifacts, and the implementation of various triggers. Such triggers are for example buttons that start and stop the time measurement for the ongoing task. Additional triggers allow us to capture the number of deletions, or the number of words for each individual language construct.
The experiment master shown in Figure 1 is aware of the formal experimental description and responsible for distributing tasks to clients and collecting all relevant metrics. After the experiment is finished, it produces according to the previous specified instructions the corresponding statistical plots and embeds them into reports (see Figure 4).
Experiment clients are used to conduct the concrete experiment, e.g., by creating test cases with DSL_1 and DSL_2 in our case. These clients are responsible for collecting relevant measures and transmitting them to the experiment master as soon as the task has been completed.

4.1.3 Analysis & interpretation

For the analysis & interpretation part, graphs are plotted and hypothesis tested according to its prior definition. For instance, in our case after performing a normality test, a t-test is performed to verify whether the time to create tests differs significantly. An example output that has been

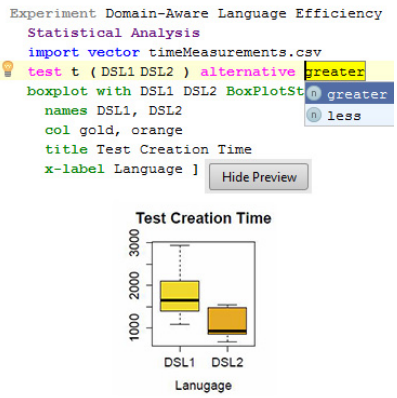

Figure 4: Screenshot of the editor

generated during the experiment as described in Section 3 is shown in Figure 4. As one might get interesting insights during analysis, it can be adapted on the fly. As an example, Figure 4 shows the box plot comparing the time taken to create the documents for both DSLs.

The resulting box-plots already indicate that the time taken to create test specifications with DSL_2 is smaller. In order to verify its significance, one has to perform further analysis. An additional t-test is performed, which is just under creation with editor support in Figure 4. Because the platform is just a very initial prototype for detailed analysis, there is also the possibility to export the collected results as comma separated version file. This allows further analysis in any other environment that supports it, for instance directly in RStudio [13] or with RapidMiner [12].

During the extended use of the platform in a real experiment we saw that automated collection of metrics as supported by our environment is important for DSL experimentation, but that also integrated support for questionnaires and their analysis is needed. For the moment this is not supported by our environment, but a top priority feature for the next milestone.

4.1.4 Presentation & Package

In order to support the entire experimentation process reports are automatically created, based on the experiment definition. At the moment our tool is capable of automatically generating PDF reports preparing information in an easy to read way. PDF reports are generated from LaTeX based on pre-filled LaTeX paper templates based on the reporting recommendations of Wohlin et al. [19] which contains all relevant information of the experiment. Presentation & packaging however, consists of more than just a PDF report. Protocols and collected data both raw and generated through the analysis should also be part of that step. At this point in time the prototype is only capable of exporting raw data and a very simple LaTex file.

5. SUMMARY & CONCLUSION

In this paper we have presented a fully integrated tool environment for experimentation in domain specific language engineering based on MPS. Our approach is to the best of our knowledge the first one providing an end-to-end support for controlled experiments in DSL engineering within a single tool environment. The environment supports all steps of experimentation (i.e. experiment planning, experiment operation, analysis & interpretation, as well as presentation & package), each supported by different specialized DSLs or extensions to the language workbench for automated metric collection. It supports the language engineer and researcher in performing experiments by providing immediate, data-driven decision support and evidence. In a running example an excerpt from a real DSL experiments with students, where an extension to a DSL for acceptance testing of web applications is validated, we show the practical application of the tool environment.

As future work, we plan to extend our tool by taking the entire feature set proposed in the ExpDSL language, which includes for instance the definition and execution of questionnaires, into account. Furthermore, we plan to conduct additional controlled experiments in academic and industrial settings to evaluate and improve the tool environment. Finally, we plan to roll out of the tool environment and its evaluation in case studies on domain language engineering in industry.

Acknowledgment

This research was partially funded by the research projects QE LaB - Living Models for Open Systems (FFG 822740) and MOBSTECO (FWF P26194).

6. REFERENCES

[1] F. Campagne. *MetaR: A DSL for statistical analysis*. Campagne Laboratory, 2015.

[2] B. Cartaxo, I. Costa, D. Abrantes, A. Santos, S. Soares, and V. Garcia. Eseml: empirical software engineering modeling language. In *Proceedings of the 2012 workshop on Domain-specific modeling*, pages 55–60. ACM, 2012.

[3] A. C. Dias-Neto and G. H. Travassos. Model-based testing approaches selection for software projects. *Information and Software Technology*, 51(11):1487 – 1504, 2009.

[4] M. Freire, P. Accioly, G. Sizílio, E. C. Neto, U. Kulesza, E. Aranha, and P. Borba. A model-driven approach to specifying and monitoring controlled experiments in software engineering. In *Product-Focused Software Process Improvement*, pages 65–79. Springer, 2013.

[5] M. Freire, U. Kulesza, E. Aranha, G. Nery, D. Costa, A. Jedlitschka, E. Campos, S. T. Acuña, and M. N. Gómez. Assessing and evolving a domain specific language for formalizing software engineering experiments: An empirical study. *International Journal of Software Engineering and Knowledge Engineering*, 24(10):1509–1531, 2014.

[6] M. A. Freire, D. A. da Costa, E. C. Neto, T. Medeiros, U. Kulesza, E. Aranha, and S. Soares. Automated support for controlled experiments in software engineering: A systematic review (S). In *The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27-29, 2013.*, pages 504–509, 2013.

[7] P. Gabriel, M. Goulao, and V. Amaral. Do software languages engineers evaluate their languages? *arXiv preprint arXiv:1109.6794*, 2011.

[8] R. E. Garcia, E. N. Höhn, E. F. Barbosa, and J. C. Maldonado. An ontology for controlled experiments on software engineering. In *SEKE*, pages 685–690, 2008.

[9] L. Hochstein, T. Nakamura, F. Shull, N. Zazworka, V. R. Basili, and M. V. Zelkowitz. An environment for conducting families of software engineering experiments. *Advances in Computers*, 74:175–200, 2008.

[10] JetBrains Team. *MPS: Meta Programming System*. Jetbrains, 2015.

[11] T. Kosar, S. Bohra, and M. Mernik. Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77–91, 2016.

[12] RapidMiner Team, Rapid-I. *RapidMiner: Predictive Analytics Platform*, 2015.

[13] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015.

[14] H. Siy and Y. Wu. An ontology to support empirical studies in software engineering. In *Computing, Engineering and Information, 2009. ICC'09. International Conference on*, pages 12–15. IEEE, 2009.

[15] D. I. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E. F. Koren, and M. Vokác. Conducting realistic experiments in software engineering. In *International Symposium on Empirical Software Engineering 2002*, pages 17–26. IEEE, 2002.

[16] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. Goal question metric (gqm) approach. *Encyclopedia of software engineering*, 2002.

[17] M. Völter. Preliminary experience of using mbeddr for developing embedded software. In *Tagungsband des Dagstuhl-Workshops*, page 73, 2014.

[18] 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.

[19] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer, 2012.