Reducing Environmental Complexity of External DSLs with Projectional Language Workbenches

Daniel S. Fleming, K.A.Hawick

School of Engineering and Computer Science, University of Hull

Objectives

Does projection editing reduce the environmental complexity of building and maintain external DSLs?

- Explore external DSL techniques for graph analysis.
- Evaluation of generative programming for DSLs.
- Evaluation of projectional editing for external DSLs.
- Implementation of a projectionally edited graph analysis DSL with a multi-architecture generation.

Introduction

Software engineering is defined as applying the ideologies of engineering to software development. This area of engineering has to take a holistic view of the environmental complexities to avoid expensive development bottlenecks. The primary environmental concerns include managing, scheduling, budgeting, maintenance, unforeseen bugs, available tools, target architectures and the end users technical literacy. (Mills, 1980).

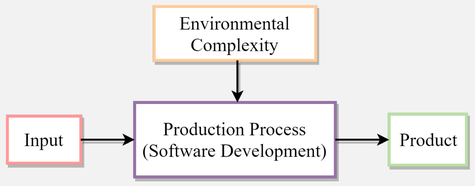

Figure 1: General production process model (Banker, 1987)

Figure 1 illustrates how the environmental complexity of software development can have a detrimental effect on the time taken to develop software. The environmental complexity is increasingly becoming a concern in relation to software bottlenecking. Where by, the growing demand for more complex software outstrips the developers. (Banker, 1987) To date, various methods have been developed to alleviate this bottleneck. The standard technique is often a divide and conquer approach through levels of abstraction such as subroutine libraries; OO frameworks and component frameworks. (Van Deursen, A., and Klint, P., 2000) Although, a relatively recent approach is to develop domain specific languages (DSLs).

Domain Specific Languages

DSLs are smaller languages targeted at a particular domain; they are a way of controlling an abstraction. Fundamentally, DSLs provide a high-level set of features that are closely aligned with the problem domain, allowing an easier mapping of the developers conceptual model to the programming implementation. DSLs are usually declarative in style.

As such, they are viewed as specification languages and programming languages. (Van Deursen, A., and Klint, P., 1998) The use of DSLs offers a broad range of possibilities for analysis, verification, high-level domain specific optimizations, parallelization, and transformation (Mernik,2005).

DSLs fall into two main categories, *internal* and *external*. Internal DSLs are built within a general purpose programming language (GPL). This method can reduce the time needed to implement the DSL though utilizing the features of the base language. Although, internal DSLs are often constrained by the host languages syntactic flexibility. Alternatively, external DSLs can use a fully custom syntax and therefore are able to be more concise and expressive. Although, there is a larger development overhead though having to build a parser, provide a programming environment, maintain the language and repeating functionality of GPLs (Fowler and Parsons, 2010).

Langauge Workbenches

Language workbenches are the set of emerging language creation environments. Their aim is to address the environmental complexity of developing external DSLs though including the following features (Fowler, M., 2017):

- Freely define new languages that can be fully integrated with each other.
- The language is stored as a persistent abstract representation.
- Language designers define a DSL in three main parts: schema, editor(s), and generator(s).
- Manipulate a DSL through a projectional editor.
- A language workbench can persist incomplete or contradictory information in its abstract representation.

Projectional Language Workbench

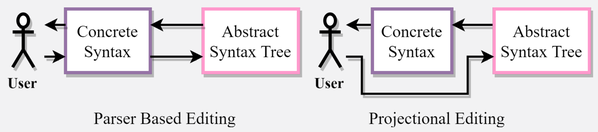

Figure 2: A comparison between a user interacting with a traditional parser based editor and projectional editor. (Voelter, M., 2014)

Many real world languages and domain areas need a mix of textual, graphical, symbolic, tabular and mathematical notations. This often isn't achievable in traditional parser based IDEs. Projectional editors are an exciting form of language workbench. The users actions directly manipulate the abstract syntax tree (AST). Therefore, they can avoid the need to utilize parsers to build an AST from a concrete syntax.

Projectional editors have four main advantages over parser based approaches (Berger, T., Völter, M., 2016):

- IDE extension
- Multi-notation composition
- Multi-language composition
- Multi-projection

This emerging technology suffers from issues of maturity. For example, at the time of writing, there is no standard format for the language storage representation, leading to problems with "vendor lock in".

		MPS	MetaEdit+	Whole	Xtext
Notation	Textural	✓	✗	✓	✓
	Graphical	✓	✓	✓	✗
	Symbols	✓	✓	✓	✗
	Tabular	✓	✓	✓	✗
	Multi-Notation	✓	✓	✓	✗
Semantics	Concrete Syntax	✓	✓	✗	✗
	Interpretive	✓	✓	✓	✓
	Model2Text	✓	✓	✓	✓
	Model2Model	✓	✓	✓	✓
Editing Type	Free-form	✗	✓	✗	✓
	Projectional	✓	✗	✓	✗
Semantic Assistance	Quick Fixes	✓	✗	✗	✓
	Reference Finding	✓	✓	✗	✓
Validation	Types	✓	✗	✗	✓

Table 1: Comparison of language workbench features. Adapted from: (Gerbig, R., 2017) (Erdweg, S., Völter, 2013)

Meta-Programming System (MPS)

MPS is an open-source projectionaly edited language workbench. It has a rich feature set to support language development and use. The structure of MPS and how the user can directly interact with the system can be seen in Figure 3. It supports language aspects for concrete and abstract syntax, type systems, constraints, transformations, plain text generation, Java template macro generation and features for IDE extension (jetbrains, 2017).

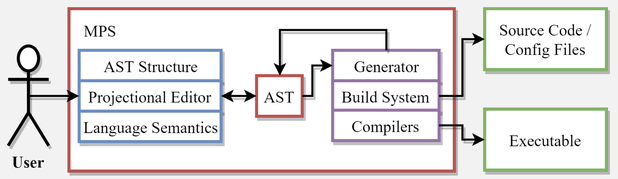

Figure 3: Structure of Jetbrains MPS (Campagne, F., 2014)

Previous research (Voelter, M., 2014) (Voelter, M. and Lisson, S., 2014) has shown that MPS avoids the significant usability issues associated with previous projectional language workbenches. This is achieved through MPS' flexible notation, language composability, editor transformations, and editor DSL. Table 1 shows a more comprehensive overview of the features of MPS as compared to other standard language workbenches.

Conclusion and Future Directions

Projectional editing is a promising emerging area of research, enabling a multitude of new ways to help bridge the gap between the conceptual model of a program and its implementation. This is achieved through more closely trailering the language design and environment to the domain itself. Future research will be focusing on experimenting and reviewing these techniques against particular domain areas. These include:

- Evaluative review of projectional editing and language workbenches.
- Implementation and evaluation of a graph analysis projectionaly edited DSL extending Green-Marl.
- Implementation and assessment of an ABM projectionaly edited DSL.

References

1) Mills, H.D., 1999. The management of software engineering, Part I: Principles of software engineering. IBM Systems Journal, 38(2.3), pp.289-295.
2) Banker, 1987, December. Factors Affecting Software Maintenance Productivity: an Exploratory Studyl. In ICIS (p. 27).
3) Van Deursen, A. and Klint, P., 1998. Little languages: little maintenance?. Journal of software maintenance, 10(2), pp.75-92.
4) Van Deursen, A., Klint, P. and Visser, J., 2000. Domain-specific languages: An annotated bibliography. ACM Sigplan Notices, 35(6), pp.26-36.
5) Mernik, M., Heering, J. and Sloane, 2005. When & how to develop domain-specific languages. ACM computing surveys (CSUR), 37(4), pp.316-344.
6) Fowler, M. and Parsons, R. (2011). Domain-specific languages. Boston, Mass.: Addison-Wesley.
7) Fowler, M. (2017). Language Workbenches: The Killer-App[online] martinfowler.com. Available at: `https://www.martinfowler.com/articles/languageWorkbench.html` [Accessed 1 Jun. 2017].
8) Berger, T., Völter, M., 2016, November. Efficiency of projectional editing: A controlled experiment. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 763-774). ACM.
9) Gerbig, R., 2017. Deep, Seamless, Multi-format, Multi-notation Definition & Use of Domain-specific Languages (Doctoral dissertation, Verlag Dr. Hut).
10) Erdweg, S., Völter, 2013, October. The state of the art in language workbenches. In International Conference on Software Language Engineering (pp. 197-217). Springer, Cham.
11) jetbrains (2017). MPS user guide for DSL users [online] Available at: `https://confluence.jetbrains.com/display/MPSD32/MPS+user+guide+for+DSL+users` [Accessed 2 Jun. 2017].
12) Campagne, F., 2014. The MPS Language Workbench: Volume I (Vol. 1). Fabien Campagne.
13) Voelter, M., 2014, September. Towards user-friendly projectional editors. In International Conference on Software Language Engineering (pp. 41-61). Springer, Cham.
14) Voelter, M. and Lisson, S., 2014, September. Supporting Diverse Notations in MPS'Projectional Editor. In GEMOC@ MoDELS (pp. 7-16).