Master project description
Projectional editing support for textual languages

Jur Bartels
Department of Computer Science and Mathematics
Technische Universiteit Eindhoven

February 19, 2019

Context

Domain-specific languages, DSLs for short, are languages designed and implemented for a specific domain and/or task. This allows language designers to focus on the concrete challenges of said domain, without having to worry about the consequences of design choices in more general use in other domains. Language workbenches exist to facilitate the creation of DSLs, often combining language design/specification, editor creation and code generation or interpretation. In this project we will concern ourselves with two language workbenches: Rascal and Jetbrain MPS.

Rascal

Rascal is a textual language workbench and meta-programming language for creating DSLs with full IDE integration. DSLs are defined by grammars, which gives it the "textual" part of the name.

MPS

MPS is a projectional language workbench. The core idea behind MPS is to decouple the language from the parser. Instead, MPS defines languages using the Abstract Syntax Tree directly. Thus to edit the language, the language programmer edits the AST by altering nodes, defining rules and constraints etc. Using the AST, MPS can create editors for the language within MPS itself in which programmers can use the language. It also provides code generation facilities for several target languages.

Project goal

The goal of this project is to research and create an interface between a textual (Rascal) and projectional (MPS) language workbench. This means that we would be able to define a language textually, through a grammar, and then use or change said language within MPS.

Approach

To be able to create an interface between both workbenches, we must first understand the differences and similarities in the languages definitions, i.e the artefacts created by the workbenches. Such artefacts for Rascal include the concrete syntax, abstract syntax, type checker, parser and interpreter. The artefacts of MPS are less known. If possible, we would want to define a possible mapping between language workbench artefacts, which may require extending one representation with additional information. Once such a mapping has been at least partially defined, we need to implement an actual interface. The implementation details, such as the location (inside one of the workbenches or independent) of this interface are to be determined later in the project, depending on the requirements posed by the mapping and the ease of implementation.

The interface will then be shown as proof of concept using a small configuration DSL of Oc.

Relevant literature

Some relevant literature on language design and workbenches:

- Lmmel, Ralf. Software Languages. Springer, Cham, 2018.
- A. Sutii, Modularity and reuse of domain-specific languages : an exploration with MetaMod Eindhoven University of Technology, 2017
- Voelter, Markus, and Konstantin Solomatov. "Language modularization and composition with projectional language workbenches illustrated with MPS." Software Language Engineering, SLE 16.3 (2010).
- Voelter, Markus, and Vaclav Pech. "Language modularity with the MPS language workbench." 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012.
- Klint, Paul, Tijs Van Der Storm, and Jurgen Vinju. "EASY Metaprogramming with Rascal." International Summer School on Generative and Transformational Techniques in Software Engineering. Springer, Berlin, Heidelberg, 2009.

Other resources

- Kogi, a project by Mauricio Verano Merino for deriving block-based environments from context-free grammars in Rascal.
- JastAdd, a meta-compilation system supporting attribute grammars.