Master Thesis outline
Projectional editing support for textual languages

Jur Bartels
Department of Computer Science and Mathematics
Technische Universiteit Eindhoven

August 28, 2019

Summary

Language composition is a very common approach used by language engineer used to combine features of two separate languages into one. In order to apply this approach however, the two languages need to exist within the same "world". We can not arbitrarily compose languages where one exists as a grammar in the textual world, and one exists as a model in the projectional world. In this project we attempt to bridge the textual and projectional worlds by creating a process for transforming textual grammars into projectional models. Language composition will then be possible by "importing" a textual language into the projectional world, and applying the composition with other projectional languages there. There are three defined sub-goals. First of all, we need to be able to map the structure of source textual grammar rules to equivalent projectional models. This step also includes the problem of how information is to be exchanged between the worlds. Secondly, we want to improve the usability of the imported language within the projectional world. There are distinct differences between the textual and projectional worlds which make languages designed for one less usable in the other. To remedy this we apply several heuristics to the imported language within the projectional world, which improve the "default" usability. Finally, after the language has been imported into the projectional world, we also want to be able to import programs written for the textual source grammar as projectional models. This removes the need to recreate already written programs within the projectional world manually. In the end, we aim to create a tool which allows for the bridging of both worlds with the least amount of manual effort required, saving the language engineer's time for more important tasks.

Chapter outlines

Abstract

Introduction

Introduce project and quickly define the problem statement and goals. Provide structure of overall thesis.

Background information

Introduce both Rascal and MPS. Explain the concepts of projectional and textual languages. Give a quick overview of both and explain the relative advantages/disadvantages

Textual languages

Short background on textual languages (grammars, parsers). Define Rascal as the example textual languages. Give a general overview of Rascal's grammar definition structures and constraints.

Projectional languages

Short background on projectional editors (model-based, AST, program by construction) Define MPS as the example projectional editor. Give a general overview of how languages are defined within MPS, i.e concepts, the different aspects (structure, editor etc.), AST nodes.

Project goal

Explain and rationalize project goal in detail. Define the problem statement and boundaries in detail. Define sub-goals: Structure, editor and programs. Explain the importance of usability in the translation of a textual grammar to MPS. Explain the program importing use case.

Related work

List all related projects and give a quick overview of their importance. Introduce Ingrid as related project with a very similar goal. (comparison to related work is further down, after our own solution has been introduced)

Solution description

General overview

Insert "world definitions" diagram so the different worlds, components and names are clearly defined. Use several cascading architectural diagrams (high to low level) Give a general overview of the solution architecture and how the different components contribute to the overall goal(s). Rationalize design choices, i.e. the XML intermediary format. Explain the approach to the problem, i.e. how the focus was on completing the full pipeline first and then improve usability.

Working example

Give a working example to be used in the next sections so each step can be explained within the context of this example. use Pico as example language and chose some interesting rules/structures to highlight.

XML

Define a specification of the intermediary XML format and rationalize it. Important to give a rigorous definition to allow other grammars to potentially be exported to it. Note limitations of the XML format.

Rascal2XML

Define the general approach of mapping a context-free grammar to XML. Explain how the Rascal to XML generation works. List any constraints placed on this generation (labeled production, unique names etc.) and rationalize them. Give concrete examples of how different constructs are mapped from a Rascal grammar definition to a XML construct for both the structure and editor aspect.

XML2MPS

Explain how XML representations of the grammars are imported in MPS. Go through the different import steps (structure and editor) in detail. Give and overview of how XML constructs are mapped to MPS structures (Concept node, interface Concept nodes etc.) and why. Give concrete examples of imported XML files. Reuse examples given above so it is clear how the full pipeline works.

Case studies

Show case studies of several source grammars imported using the process. To include: State machines, Pico, Rascal, Java and the Oc case. Comment on the results/problems of each case.

Evaluation of solution

Restate problem statement and sub-goals. Explain how the described solution steps solve the sub-goals and how the overall solution answers the problem statement. List any problems and/or limitations the solution might still have with concrete examples. Use case studies (Pico, Rascal, Java, Oc example) Give a comparison to Ingrid/other related works as to the different features, advantages and disadvantages this work has to them.

Possible improvements

Give possible solutions to problems/limitations stated in the previous section. If none are know/feasible explain why. List possible future extensions that could be implemented such as exports from different grammars to XML.

Conclusion

Restate the problem statement and how this work attempts to solve it.