ScheMoose - Supporting a Functional Language in Moose

Katerina Barone-Adesi *and* Michele Lanza
Faculty of Informatics - University of Lugano, Switzerland

Abstract

The Moose Reengineering environment has traditionally been targeted at object-oriented languages with little to no support for other paradigms, although this is theoretically supported too. In this article we describe the experiences we obtained while implementing ScheMoose, a prototype parser and MSE exporter of Scheme code written itself in Scheme. We describe to what extent ScheMoose could make use of the FAMIX metamodel, in which cases we needed to extend the meta-model and also provide a validation of ScheMoose in the form of Mondrian visualizations of Scheme programs.

1 Introduction

Recently, functional programming languages such as Scheme and LISP are being reconsidered, due to their elegance and the abstraction capabilities that they offer. The simplicity of a functional language like Scheme makes it an ideal tool to teach the fundamentals of programming, as we do ourselves at our university.

The ScheMoose project's motivation lies in the many systems that we are being delivered by the students and by the lack of comprehension tools that are available for Scheme. Our goal is to reuse the existing infrastructure of the Moose reengineering environment [1], by extending it with support for functional languages. This involves the following steps, which also roughly dictate the structure of the paper:

- The creation of a parser for Scheme code written in Scheme and the implementation an exporter in Scheme which exports the modeled code to the mse format
- The subsequent modeling within Moose and the necessary extensions/additions to Moose to better cope with functional source code
- The final validation of ScheMoose by means of visualizations obtained using the Mondrian framework [2].

2 ScheMoose

Schemoose is a Scheme program which extracts information from Scheme source code and exports it as an mse file, using the FAMIX metamodel. It models functions and variable definitions, project structure (i.e., which definitions are in which file), and function calls. It also determines some semantic information about the code, such as whether a function is a predicate function.

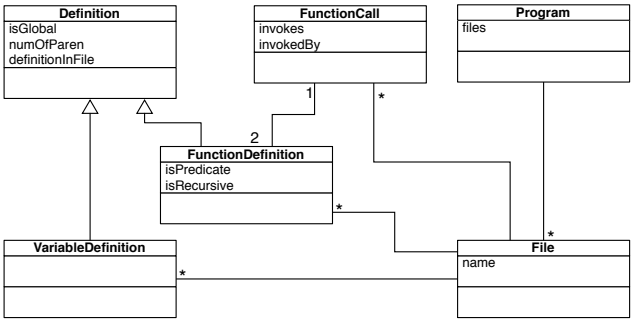

Figure 1. ScheMoose metamodel

We are currently adding semantic information to the extracted model, such as whether a function is directly recursive (it calls itself) or is mutually recursive with another functions, etc.

In Figure 1 we see a reduced class diagram of the information we currently model. We expect to soon finalize the amount of information that we extract from Scheme code. In terms of metrics we currently defined the number of parentheses (NPAR) and the position within the file (DIF). While the first metric is concerned with modeling the complexity of the function, we plan to exploit the DIF metric to obtain a sequential view of the code, since evaluation order in Scheme is a primary concern, and also reveals information about the way the code has been written.

ScheMoose, itself written in Scheme, makes use of the Scheme function `read` to extract the information.

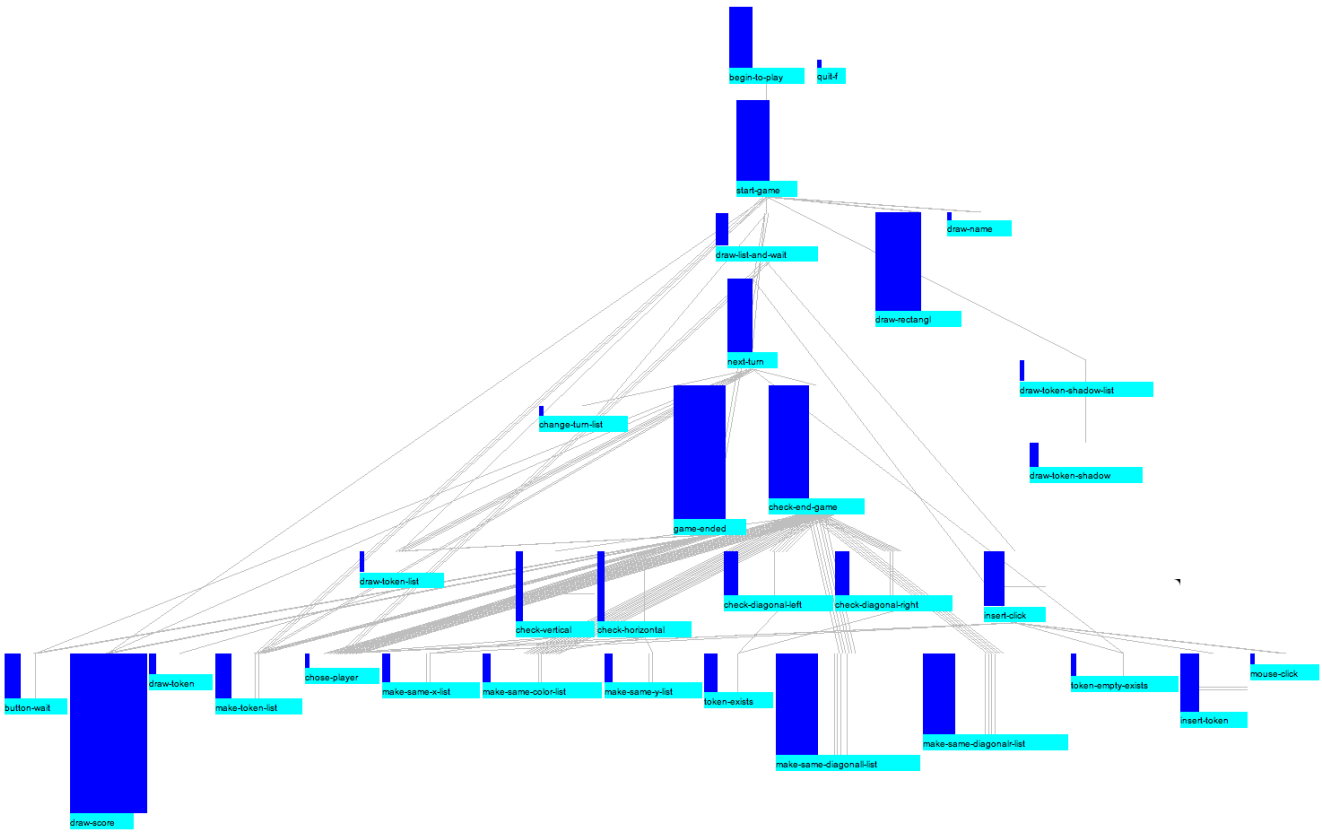

Figure 2. Mondrian visualization of a Connect-4 game written in Scheme

3 Enriching FAMIX

To a large extent Moose if already apt at handling functional languages, i.e., in our specific case we can reuse the existing FAMIXFunction and FAMIXInvocation classes.

ScheMoose is work in progress, and because of that we only extended FAMIXFunction with the meta-description of "isPredicate", "numberOfParentheses", and "isDirectlyRecursive" so far.

4 Applications

We used Mondrian to visualize the code of one of the student projects written during the course "Programming Fundamentals".

In Figure 2 we see the whole program displayed as a tree of functions that invoke each other. The boxes display information about each function, namely the number of outgoing invocations for the width, and the number of parentheses for the height.

5 Conclusions

Our preliminary experiences, since the ScheMoose project is still going on, is that Moose is fit for modeling functional languages, and that the necessary extensions to some of the FAMIX classes can be considered marginal.

There are still a number of things we plan to implement, such as supporting macros, higher-order functions, continuations, conditional expressions, etc.

References

[1] S. Ducasse, T. Gîrba, M. Lanza, and S. Demeyer. Moose: a collaborative and extensible reengineering environment. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 55–71. Franco Angeli, Milano, 2005.

[2] M. Meyer, T. Gîrba, and M. Lungu. Mondrian: An agile visualization framework. In *ACM Symposium on Software Visualization (SoftVis 2006)*, pages 135–144, New York, NY, USA, 2006. ACM Press. To appear.