Chapter 4

An Evaluation of a Pure Embedded Domain-Specific Language for Strategic Term Rewriting

Shirren Premaratne
Macquarie University, Australia

Anthony M. Sloane
Macquarie University, Australia

Leonard G. C. Hamey
Macquarie University, Australia

ABSTRACT

Domain-specific languages are often implemented by embedding them in general-purpose programming languages. The Kiama library used in this chapter for the Scala programming language contains a rewriting component that is an embedded implementation of the Stratego term rewriting language. The authors evaluate the trade-offs inherent in this approach and its practicality via a non-trivial case study. An existing Stratego implementation of a compiler for the Apply image processing language was translated into a Kiama implementation. The chapter examines the linguistic differences between the two versions of the Stratego domain-specific language, and compares the size, speed, and memory usage of the two Apply compiler implementations. The authors’ experience shows that the embedded language implementation inflicts constraints that mean a precise duplication of Stratego is impossible, but the main flavor of the language is preserved. The implementation approach allows for writing code of similar size, but imposes a performance penalty. Nevertheless, the performance is still at a practically useful level and scales for large inputs in the same way as the Stratego implementation.
INTRODUCTION

One popular domain-specific language (DSL) implementation approach is to embed the DSL in a general-purpose host language to create an internal language (Mernik, 2005; Fowler, 2010; Ghosh, 2011). In this chapter we consider internal languages where the embedding consists only of pure DSL constructs that are written directly using the host language and an unchanged host language compiler performs the only compilation or translation step. In essence, the DSL is a host language library but the syntactic flexibility of the host language is exploited to implement the DSL syntax.

The embedding approach is attractive for a number of reasons, but has associated drawbacks. Reusing the host language compiler significantly simplifies implementation of the DSL compared to implementing a standalone version of the language — all host language tools can be reused. The main drawback is that the DSL syntax and semantics may not be directly realizable in the host language, resulting in compromises. Host language tools may not present a domain-specific view of programs, requiring the programmer to be aware of the way in which the DSL is implemented. Nevertheless, the pure embedding approach can be useful, particularly where the target users are developers who are familiar with the host language and where there is resistance to the adoption of new tools or build processes.

How can we evaluate the embedding approach beyond these generic high-level considerations? Our view is that real insight can only be gained through case studies of DSLs of varying styles and domains. In particular, if an external DSL has already been designed and a standalone implementation exists, then a side-by-side comparison with a new embedded version can be especially revealing. Evaluating how close the internal implementation gets to the existing external implementation provides a measure of the success of this approach. It is not our intention to undertake a full comparison of the implementation techniques, but to treat the existing external implementation as an ideal and to evaluate the ability of the embedded approach to reproduce that ideal.

Recently we have completed a project that studied an embedded implementation of a strategic term rewriting DSL and compared it to an existing implementation (Premaratne, 2011). (The code from the project is available at https://wiki.mq.edu.au/display/plrg/stragma.) The Kiama library (Sloane, 2011) contains term rewriting features inspired by an existing external language called Stratego (Visser, 2004). Kiama is embedded in the Scala general-purpose language (Odersky, 2008), which was chosen for its excellent support of pattern matching, flexible syntax and powerful static type system. When we developed the rewriting component of Kiama we adhered to the Stratego design as much as was possible and sensible, given the constraints and opportunities provided by Scala.

In this project, Stratego played the role of a well-established external DSL that is implemented by a compiler and Kiama played the role of an internal implementation of that DSL. We evaluated the two implementations using a non-trivial Stratego application that one of us had developed previously: a compiler for the Apply image processing language (Hamey, Webb & Wu, 1987; Hamey, 2007; Hamey & Goldrei, 2008). The Apply compiler translates Apply programs into C, conducting standard semantic analysis and non-trivial optimization along the way. We translated the existing Stratego implementation of the Apply compiler into Scala using the Kiama library. The two implementations perform the same analyses and translations and both run on the Java Virtual Machine. Since the focus of the comparison was the rewriting DSL, the two implementations of Apply share a common implementation of other passes such as parsing and pretty printing.

This chapter presents the results of the project with a focus on the trade-offs that the embedding approach imposes on the rewriting language and