A Little Language for Surveys: Constructing an Internal DSL in Ruby
H. Conrad Cunningham

Hudak defines a *domain-specific language* (DSL) as "a programming language tailored to a particular application domain" [10]. It is a *little language* [1] that sacrifices generality to obtain expressiveness in the targeted domain [11]. A DSL should enable experts in an application area to program without programming—that is, to express the problems they want the computer to solve using familiar concepts and notations, without having to master the intricacies of programming in a general-purpose language [10,16]. For example, the DSL `pic` (long available on Unix-based computers) enables writers to produce line drawings in documents; they can focus on the layout of the drawings without being required to develop programs in C (the primary general-purpose language used on Unix) [1].

The designers of a DSL must select relevant concepts, notations, and processes from the application domain and incorporate them into the DSL design [10]. Often they approach this task in an ad hoc manner. A goal of our research is to identify ways to approach DSL design systematically. In this paper, we leverage commonality-variability analysis [4] of a domain to help identify the needed language constructs and their semantics; we exploit software design patterns [9] and DSL patterns [6] to introduce the desired variability into the DSL's implementation.

Fowler classifies DSLs into two styles—external and internal [6]. Although the terminology is relatively new, the ideas are not. An *external DSL* is a language that is different from the main programming language for an application, but that is interpreted by or translated into a program in the main language. The language `pic` exhibits this style. An *internal DSL* transforms the main programming language itself into the DSL—the DSL is *embedded* in the main language [10]. The language Lisp (which was defined in the 1960s) supports syntactic macros, a convenient mechanism for extending the language by adding application-specific features.

The Ruby programming language [14]—with its flexible syntax and extensive reflexive metaprogramming facilities [3]—provides a convenient platform for developing internal DSLs. The rise in popularity of Ruby and the associated Ruby on Rails web framework [15] has stimulated new interest in DSLs among practitioners [2,8].

This paper takes a problem motivated by Bentley's classic column on "Little Languages" [1], constructing a little language for surveys, explores the DSL capabilities of the Ruby language, and designs an internal DSL for specifying and executing surveys. Section 2 describes the Ruby facilities for constructing internal DSLs. Section 3 analyzes the survey problem domain and designs a simple DSL based on the analysis. Section 4 sketches the design and implementation of the survey DSL processor. Sections 5 and 6 examine this work from a broader perspective and conclude the paper.

REFERENCES

H. C. Cunningham. A little language for surveys: Constructing an internal DSL in Ruby, In *Proceedings of the ACM SouthEast Conference*, 6 pages, March 2008.

For this submission I rewrote the Introduction to a paper I published in 2008. My rewrite uses the same citations with the same numbering I used in the original. I replaced item 6 (a web link to work in progress) with the book the author later published from the material I cited.

1. J. Bentley. Programming pearls: Little languages. *Communications of the ACM*, 29(8):711–721, August 1986.
2. J. Buck. Writing domain specific languages. http://weblog.jamisbuck.org, April 2006.
3. L. Carlson and L. Richardson. *Ruby Cookbook*. O'Reilly, 2006.
4. J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *IEEE Software*, 15(6):37–45, November 1998.
5. M. Fowler. Using the `rake` build language. http://martinfowler.com/articles/rake.html, August 2005.
6. M. Fowler. *Domain Specific Languages*. Addison Wesley, 2011.
7. S. Freeman and N. Pryce. Evolving an embedded domain-specific language in Java. In *Companion to the Conference on Object-Oriented Programming Languages, Systems, and Applications*, pages 855–865. ACM SIGPLAN, October 2006.
8. J. Freeze. Creating DSLs with Ruby. *Artima Developer: Ruby Code and Style*, March 2006. http://www.artima.com/rubycs/articles/ruby_as_dsl.html.
9. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
10. P. Hudak. Modular domain specific languages and Tools. In P. Devanbu and J. Poulin, editors, *Proceeding of the 5th International Conference on Software Reuse (ICSR'98)*, pages 134–142. IEEE, 1998.
11. M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain specific languages. ACM Computing Surveys, 37(4):316–344, December 2005.
12. T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.
13. S. Thibault, R. Marlet, and C. Consel. Domain-specific languages: From design to implementation—Application to video device driver generation. *IEEE Transactions on Software Engineering*, 25(3):363–377, May/June 1999.
14. D. Thomas, C. Fowler, and A. Hunt. *Programming Ruby: The Pragmatic Programmers' Guide*. Pragmatic Bookshelf, second edition, 2005.
15. D. Thomas and D. Heinemeier Hansson. *Agile Development with Rails.* Pragmatic Bookshelf, second edition, 2006.
16. A. van Deursen, P. Klint, and J. Visser. Domain specific languages: An annotated bibliography. *SIGPLAN Notices*, 35(6):26–36, June 2000.
17. *Wikipedia, The Free Encyclopedia. Metaprogramming*. http://en.wikipedia.org/wiki/Metaprogramming, Accessed 22 February 2008.