Hudak defines a *domain-specific language* (or DSL) as "a programming language tailored to a particular application domain" [10]. DSLs are usually not general-purpose languages; they instead "trade generality for expressiveness in a limited domain" [11]. Thus DSLs must be precise in capturing the semantics of their application areas [10]. They are usually small, declarative languages targeted at end users or domain specialists who are not expert programmers [10, 16].

DSLs, often known as *little languages*, have long been important in the Unix operating systems community. For example, in an influential 1986 column [1], Bentley describes the little line-drawing language `pic` and its preprocessors `scatter` (a language for drawing scatter plots of two-dimensional data) and `chem` (a language for drawing molecular structures). He also describes other well-known little languages that are used to implement `pic`: `lex` for specifying lexical analyzers, `yacc` for specifying parsers, and `make` for specifying build processes.

Fowler classifies DSLs into two styles—external and internal [6]. 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 little languages from the Unix platform are in this category. The document preparation languages LATEX and BibTeX, which the author is using to format this paper, are also external DSLs. External DSLs may use ad hoc techniques, such as hand-coded delimiter-directed or recursive descent parsers [6], or may use parser-generation tools such as `lex` and `yacc` or ANTLR [12].

What Fowler calls an *internal DSL* (and Hudak calls a domain-specific *embedded* language [10]) transforms the main programming language itself into the DSL. This is not a new idea; usage of syntactic macros has been a part of the Lisp tradition for several decades. However, the features of a few contemporary languages offer new opportunities for constructing internal DSLs.

The rise in popularity of the Ruby programming language [14] and the associated Ruby on Rails web framework [15] has simulated new interest in DSLs among practitioners. In the Ruby environment, there is significant interest in developing internal DSLs [2, 8] that are made possible by the extensive reflexive metaprogramming facilities of Ruby [3]. One interesting language of this nature is `rake`, a build language implemented as a DSL in Ruby [5].

This paper takes a problem motivated by Bentley's "Little Languages" column [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

[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. http://martinfowler.com/dslwip/, Work in progress 2007.
[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.