Fowler's Lair DSL

H. Conrad Cunningham

16 April 2022

Contents

Professor of Computer and Information Science
University of Mississippi
214 Weir Hall
P.O. Box 1848
University, MS 38677
(662) 915-7396 (dept. office)

Browser Advisory: The HTML version of this textbook requires a browser that supports the display of MathML. A good choice as of April 2022 is a recent version of Firefox from Mozilla.

Fowler's Lair DSL

Background

The Lair DSL case study is based on Martin Fowler's One Lair and Twenty Ruby DSLs, Chapter 3, in *The ThoughtWorks Anthology: Essays on Software Technology and Innovation*, Pragmatic Bookshelf, 2008 [1].

- local copy of Lair chapter
- list of DSL Patterns from Martin Fowler's *Domain Specific Languages* [2]

Source Code

The original DSLs were developed in Ruby [11,12] by Martin Fowler. All of Fowler's source code is in this folder or in file LairDSLsRuby.zip.

Fowler elaborates his patterns-based approach to domain-specific languages in his 2010 book by that name [2].

The Lua [5,8] and Python [9,10] versions were developed by H. Conrad Cunningham.

Shared modules

Fowler's Ruby (2008)

- Semantic model (model.rb)

- Test Driver for semantic model (rules0.rb)

Lua (2013)

- Class support module (for implementing classes in Lua)
- Semantic model
- Test driver for semantic model (rules00.lua)

Python (2018)

- Semantic model
- Test driver for semantic model (rules00.py)

Internal DSL using Global Function Sequence pattern

Fowler's Ruby

- builder module (builder8.rb)
- dsl script (rules8.rb)

Lua (2013)

- builder module (builder08.lua)
- dsl script (rules08.lua)
- test driver (test08.lua)

Python (2018)

- builder module (builder08.py)
- direct execution test of DSL script (rules08x.py)
- dynamically loaded dsl script (rules08.py)
- test driver for dynamically loaded dsl script (test08.py)

Internal DSL using Class Method Function Sequence and Method Chaining patterns

Fowler's Ruby

- builder module (builder11.rb)
- dsl script (rules11.rb)

Lua (2013)

- builder module (builder11.lua)
- dsl script (rules11.lua)
- test driver (test11.lua)

Python (2018)

- builder module (builder11.py)
- dynamically loaded dsl script (rules11.py)
- test driver for dynamically loaded dsl script (test11.py)

Internal DSL using Expression Builder and Method Chaining patterns

Fowler's Ruby

- builder module (builder14.rb)
- dsl script (rules14.rb)

Lua (2013)

- builder module (builder14.lua)
- dsl script (rules14.lua)
- test driver (test14.lua)

Python (2018)

- builder module (builder14.py)
- dynamically loaded dsl script (rules14.py)
- test driver for dynamically loaded dsl script (test14.py)

Internal DSL using Nested Closures pattern

Fowler's Ruby

- builder module (builder3.rb)
- dsl script (rules3.rb)

Lua (2013)

- builder module (builder03.lua)
- dsl script (rules03.lua)
- test driver (test03.lua)

Python Python's weak syntactic support for lambdas does not allow the relatively direct approach to be used as in Ruby, Scala, and Lua.

Internal DSL using Expression Builder, Object Scoping, and Method Chaining patterns

Fowler's Ruby

- builder module (builder17.4b)
- dsl script (rules17.rb)

Lua (2013)

- builder module (builder17.lua)
- dsl script (rules17.lua)
- test driver (test17.lua)

Python I have not yet developed a Python Lair DSL using these techniques.

Internal DSL using Literal Collection pattern

Fowler's Ruby

- builder module (builder22.rb)
- dsl script (rules22.rb)

Lua (2013)

- builder module (builder22.lua)
- dsl script (rules22.lua)
- test driver (test22.lua)

Python I have not yet developed a Python Lair DSL using these techniques.

External DSL using Parser/Builder

Fowler's Ruby There is no corresponding example in Fowler's chapter.

Lua (2013) Note: These programs require the Lua LPEG library [Ierusalimschy2009; [7]], which can be installed via luarocks. The library version must be compatible with whatever version of Lua is being used.

- builder module (builderLPEG1.lua)
- dsl script (rulesLPEG1.dsl)

- test driver (testLPEG1.lua)

Python (2018) Note: These programs require the Python package Parsita [3], a parser combinator library similar to Scala's.

- builder module (builderParsita1.py)
- dsl script (rulesParsita1.dsl)
- test driver (testParsita1.py)

Acknowldgments

I thank Martin Fowler for developing the interesting Lair case study and DSL examples [1] and for writing his book on domain-specific languages [2]. I thank Fowler, Thoughtworks, and Pragmatic Bookshelf for making the chapter and source code available.

I developed the Lua versions for my CSci 658 (Software Language Engineering) course in Fall 2013. I thank Roberto Ierusalimschy and the LabLua team at PUC-Rio for developing the interesting minimalistic Lua language [8], LPEG parsing library [4,7], and other resources. I also thank Ierusalimschy for writing the helpful book *Programing in Lua* [6].

I developed the Python versions for my CSci 658 course in Spring 2018. I thank those who support Python and its extensive ecosystem. I also think David Hagen for developing the parsing combinator library Parsita [3].

I thank the students in the Scala-, Lua-, and Python-based versions of the Software Language Engineering course for their patience with sometimes immature examples and for their feedback.

I retired from the full-time faculty in May 2019. As one of my post-retirement projects, I am continuing work on textbooks related to the courses I taught during myt 30 years as a computer science faculty member. In Spring 2022, I began refining the existing content, integrating additional separately developed materials, reformatting the documents (e.g., using CSS), constructing a unified bibliography (e.g., using citeproc), and improving the build workflow and use of Pandoc. I adapted this index page from a portion of my Spring 2018 course notes.

I maintain this chapter as text in Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the document to HTML, PDF, and other forms as needed.

References

[1] Martin Fowler. 2008. One lair and twenty Ruby DSLs. In *The ThoughtWorks anthology: Essays on software technology and innovation*, ThoughtWorks, Inc. (ed.). Pragmatic Bookshelf, Raleigh, North Carolina, USA. Retrieved from https://media.pragprog.com/titles/twa/martin_fowler.pdf

[2] Martin Fowler and Rebecca Parsons. 2010. *Domain specific languages*. Addison-Wesley, Boston, Massachusetts, USA.

[3] David Hagen. 2022. Parsita. Retrieved from https://pypi.org/project/parsita/

[4] Roberto Ierusalimschy. 2009. A text pattern-matching tool based on parsing expression grammars. *Software: Practice and Experience* 39, 3 (2009), 221–258. Retrieved from http://www.inf.puc-rio.br/~roberto/lpeg/lpeg.html

[5] Roberto Ierusalimschy. 2013. *Programming in Lua* (Third ed.). Lua.org, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil.

[6] Roberto Ierusalimschy. 2016. *Programming in Lua* (Fourth ed.). Lua.org, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil.

[7] Roberto Ierusalimschy. 2022. LPeg: Parsing expression grammars for Lua, version 1.0. Retrieved from http://www.inf.puc-rio.br/~roberto/lpeg/

[8] LabLua, PUC-Rio. 2022. Lua: The programming language. Retrieved from https://www.lua.org/

[9] Python Software Foundation. 2022. Python. Retrieved from https://www.python.org/

[10] Luciano Ramalho. 2013. *Fluent Python: Clear, concise, and effective programming*. O'Reilly Media, Sebastopol, California, USA.

[11] Ruby Community. 2022. Ruby: A programmer's best friend. Retrieved from https://www.ruby-lang.org

[12] David Thomas, Chad Fowler, and Andrew Hunt. 2004. *Programming Ruby* (Second ed.). Pragmatic Bookshelf, Raleigh, North Carolina, USA.