CSci 658: Software Language Engineering
Sandwich DSL Case Study (Lua)

H. Conrad Cunningham

3 March 2018

Contents

Sandwich DSL Case Study **1**
Problem Description 1
Tasks 2
One Lua Solution 3
Lua Data Representation 3
Lua Internal DSL 3
Testing 4
Acknowledgements 4
References 4
Concepts 4

Professor of Computer and Information Science
University of Mississippi
211 Weir Hall
P.O. Box 1848
University, MS 38677
(662) 915-5358

Advisory: The HTML version of this document requires use of a browser that supports the display of MathML. A good choice as of March 2018 is a recent version of Firefox from Mozilla.

Sandwich DSL Case Study

Problem Description

Suppose Emerald de Gassy, the owner of the Oxford-based catering business Deli-Gate, hires us to design a domain-specific language (DSL) for describing sandwich platters. The DSL scripts will direct Deli-Gate's robotic kitchen appliance SueChef (Sandwich and Utility Electronic Chef) to assemble platters of sandwiches.

In discussing the problem with Emerald and the Deli-Gate staff, we discover the following:

- A sandwich platter consists of zero or more sandwiches. (Zero? Why not! Although a platter with no sandwiches may not be a useful, or profitable, case, there does not seem to be any harm in allowing this degenerate case. It may simplify some of the coding and representation.)
- Each sandwich consists of layers of ingredients.
- The categories of ingredients are breads, meats, cheeses, vegetables, and condiments.
- Available breads are white, wheat, and rye.
- Available meats are turkey, chicken, ham, roast beef, and tofu. (Okay, tofu is not a meat, but it is a good protein source for those who do not wish to eat meat. This is a college town after all. Oh, there is also a special meat served for football games Thanksgiving week called "bulldog", but it is really just chicken, so we can ignore that choice for our purposes here.)
- Available cheeses are American, Swiss, jack, and cheddar.
- Available vegetables are tomato, lettuce, onion, and bell pepper.
- Available condiments are mayo, mustard, relish, and Tabasco. (Of course, this being the South, the mayo is Blue Plate Mayonnaise and the mustard is a Creole mustard.) Creole mustard.)
- The sandwiches must be assembled in Oxford Standard Order (OS slice of bread on the bottom, then zero or more meats layered above that, then zero or more cheeses, then zero or more vegetables, then zero or more condiments, and then a slice of bread on top. The top and bottom slices of bread must be of the same type. (OSO was defined by the mysterious and picky food critic known as CH.)

Tasks

1. Describe an appropriate semantic model for your Lua Sandwich DSL. That is, what concrete data structures might you use? What functions/methods to manipulate the semantic model?
2. Remember the sandwiches must be assembled in OSO. Choose to do either an external DSL or an internal DSL. Tell me which. Describe the syntax for your DSL.

3. **Give a Lua script for configuring the following platter:** one turkey and Swiss on rye with tomatoes and mayo; one tofu and mustard on wheat.

4. Describe a design for an implementation for your DSL. That is, how would you translate a DSL script into an appropriate configuration of the semantic model? What DSL implementation techniques or tools would you use? Be specific.

One Lua Solution

Lua Data Representation

This semantic model design and implementation (in Lua module `sandwich_model.lua`) uses Lua tables to represent platters and sandwiches. (This representation is more general than needed for the Sandwich DSL problem from the exam.)

A list-style Lua table

```
{ PLATTER, n, sandwich1, sandwich2, ... sandwichM }
```

describes a set of `n` identical platters, for some `n > 0`. `PLATTER` is a tag constant. Each platter has `M` descriptions of sandwiches, for some `M >= 0`.

A list-style table

```
{ SANDWICH, n, layer1, layer2, ... layerM }
```

describes a set of `n` identical sandwiches, for some `n > 0`. Each sandwich has `M` layers of ingredients ordered from bottom to top, for some `M >= 0`.

Unlike the Lair semantic model, this semantic model is not technically object-oriented. However, the tables described above are objects constructed and manipulated by the functions in this module. A DSL builder module should not create or access these directly. But more functions may be needed to enable effective use of this model by other tools.

Lua Internal DSL

The Lua module `sandwich_builder.lua` implements an internal Sandwich DSL in Lua. It uses Fowler's Function Sequence and Context Variables DSL patterns. The syntax has the structure:

```
platter()
sandwich()
    ingredient commands
sandwich()
    ...
end_platter()
```

The ingredient commands consist of the following in any order, but without repetition within a sandwich:

```
bread(b)
meats(m1,m2,...)
cheeses(c1,c2,...)
vegetables(v1,v2,...)
condiments(c1,c2,...)
```

Note that only one bread may be specified, but one or more of the other ingredients may be specified in a command.

A `bread` must be specified, but all the other commands are only given if ingredients of the associated type are to be added to the sandwich. Regardless of the order of the commands, the sandwich is constructed in Oxford Standard Order – bread, meats, cheeses, vegetables, condiments, bread from bottom to top.

The call `get_platter()` extracts the platter from the builder.

Testing

The Lua module `test_sandwichDS.lua` implements a simple test run of this internal DSL.

Acknowledgements

The problem described in this document was an extended exercise on an examination in the Lua-based CSci 658 course in Fall 2013. The text above is extracted from the comments at the beginning of my solution files.

In Spring 2018, I created this Pandoc Markdown document from the source code comments for my solutions.

This problem motivated the related Haskell SandwichDSL case study defined in Fall 2014 for CSci 450 (and then updated for Fall 2017). That case study, in turn, was reinterpreted as Scala case study for the Spring 2016 CSci 555 class. I updated the descriptions of these case studies in Spring 2018.

I maintain these notes as text in Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the notes to HTML, PDF, and other forms as needed. The HTML version of this document may require use of a browser that supports the display of MathML.

References

TODO

Concepts

TODO

Internal DSL; DSL patterns Semantic Model, Function Sequence, and Context Variable; builder