Using Lua to Build Domain Specific Languages

Using Lua to Build Domain Specific Languages

Tom Wrensch, University of the Pacific
Stockton, California, USA

Domain-Specific Languages (DSLs)

- Language design around concepts and structures of problem domain
- Operates at level of abstraction of domain
  - "Linguistic Level of Analysis"
- Focus on set of tasks or describe set of objects from domain.
- Examples: Spreadsheets, grep, SQL,...
- Usually designed for non-technical users.
- General purpose programming capabilities?
- DSLs empower users:
  - Uses their terminology and ideas.
  - High-level of abstraction (so small amount of code goes a long way)
  - "Get the job done" design philosophy

Domain-Oriented Programming (DOP) Languages

- Programming Languages that:

  > Offer one or more strong computational metaphors that allow domain concepts to be readily modeled. DOP languages have simple syntax...They often achieve their expressiveness by a strict uniformity of operations and types. DOP languages allow the domain developer to map domain abstractions into DOP abstractions. - Thomas and Barry (OOPSLA'03)

- Examples: Lisp, APL, Smalltalk, and (of course)... Lua.

Example 1: Student Grading Using Lua Syntax

- The easiest (and usually best!) way to build a DSL with Lua is to use Lua's syntax and some custom code.
- Grades for students: programs I had didn't do what I wanted.
  - Concepts: student, Course, Assignment, Group, Grade
  - Knowledge Engineering is important in DSLs, even simple ones.
    - Grade versus GradeSet problem.

Example 1: Student Grading Using Lua Syntax

```
Student ("HFred", "Fred Hamster", "f_hamster@pacific.edu")
...
Course("Comp141", "Programming Languages")
Group("Homework", 40)
Group("Tests", 40)
Group("Project" 20)
Assignment {name="HW01", desc="Lexical Analyzer",
            due="23 Sept", points=20}

GradeSet {
{"COMP141", "HW01",  "HW02", }
--   -------    ----     ----
{"HFred",      19,      10,  }
{"CMiko",      17,      12,  }
...
}
```

Example 2: Poetry Composer Using gsub

- Useful technique for bridging existing languages or other special cases.
  - Powerful and fast but also difficult to write/read
  - Hard to do good error messages.
- Used to read and existing description system for words and poem patterns.
  - Concepts: Word, Syllable count, Concept, Rhyme, Phrase, Poem
  - Structures: concept groups, concept matching, phrase, poem
- Implementation Issues: Break into bite-sized pieces first.
  - Declarations, statements, lines, whatever works.
- Iterative application of gsub is as powerful as a grammar-based system.

Example 2: Poetry Composer Using gsub

```
--- Words
tree      1 noun xee
crushing  2 ing xing
coding    2 ing verb xing
-- Concepts/Ideas
POS -> noun adjective
adjective -> ing
RHYME -> xee xing
-- Poem
: POEM haikulike
: 5 (noun) (ing verb) (noun)
: 7 (noun) (verb) (prep) (article) (noun)
: 5 (noun) (verb) *
: END
: COMPOSE 100 haikulike
```

Example 2: Poetry Composer Using gsub

Lua writing starts
Fingers fly across the keys
Elegance lives on

Example 3: Research Prototype Using Parser

- Traditional language implementation works pretty well too.
  - Lexical scanner, Parser (Top-down), Object-based parse tree.
- Prototype of high-level class-focused language.
  - Domain is object-oriented design.
  - Code defines how classes interact, not objects and methods.
  - Concepts: class, list, value, address/path, others?
  - Structures: subclassing, tree-like class structure, paths.

Example 3: Research Prototype using Parser

```
define ParentChild($parent, $plist, $child, $parref)
List($parent.$children)
Value($child.$parref)
Action($child.$parref.onUnset,apply(self.value $plist),
.remove, self)
Action($child.$parref.onSet,apply(self.value,$plist),
.add, self)
end

class(Book)
class(Shelf)
List(Book.authors, String)
Value(Book.title, String)
Value(Book.location, Shelf)
ParentChild(Shelf.contents, Book.location)
```

Conclusions

- DSLs are good.
- DOP Languages are the best choice for building DSLs.
- Lua is a DOP Language
- Know the Audience, know the domain.
- Start with Lua syntax for prototyping, stay with it if possible!
- ...but Lua can take you farther if you need it.