The evolution of Lua

Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes

October 8, 2008

Roberto Ierusalimschy

- Associate Professor, Departamento de Informtica, PUC-Rio
- CS background

Luiz Henrique de Figueiredo

- Associate Researcher at Instituto Nacional De Matematica Pura E Aplicada
- Maths background

Waldemar Celes

- Assistant Professor, Departamento de Informtica, PUC-Rio
- Associate Research at Tecgraf/PUC-Rio
- Engineering background

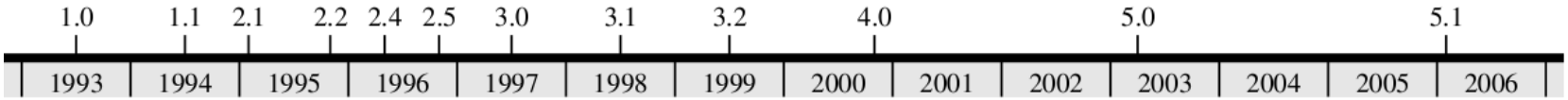
Birth of Lua

- Meant to replace two languages used by Petrobras
  - Simple declarative language
  - Specialized description language
- Wanted: Imperative, with good data-description facilities, simple syntax, portability and a C API
- In 1993, only real possibility was Tcl
- $\Rightarrow$ Lua 1.0 created

The fundamentals

- *Extension language*, because it can be embedded into other applications (written in e.g. C)
- *Extensible language*, because it has a type to hold application data and extensible semantics to manipulate these values
- portability
- performance

The basic features



- garbage collection
- extensible semantics (since 2.1)
- OOP support (since 2.1)
- external compiler (since 2.4)
- closures (since 3.2)
- multi-state API (4.0)
- full lexical scoping (5.0)
- coroutines (5.0)
- module system (5.1)

The evolution

"We will now discuss in a little more detail the Struggle for Existence."
Charles Darwin, Origin Of Species, 1859

Tables

- Lua calls associative arrays “tables”
- Lua 2.1 allowed mixed constructor
- Semantics never changed
- Implementation did though: 5.0 introduced hybrid representation (hash part, array part)
- Since 5.1 the module/package system also is based on tables
- Global variables are stored in a table and 5.1 lets the programmer store the entire environment (methods, C functions, userdata) in a table

Strings and comments

- Strings have been supported in Lua since 1.0
- Support for nested block comments since 5.1

Lexical scoping

- AKA static scoping, is a nice-to-have for programming languages because the programmer can reason better about the code
- Lua 3.1: *upvalues* – not the real deal though
- Full support in Lua 5.0 (indirection, keeping list of open *upvalues*, moving them to a heap once they get out of scope)

Coroutines

- Coroutines/continuations are functions that can *yield* and *resume*)
- Difficult to implement for C calls
- Lua 5.0 introduced coroutines with a limitation

Extensible semantics

- Lua 2.1: *fallbacks* (think: resumable exception handling)
- using fallbacks inheritance can be implemented (define a fallback that sends unknown calls to a predecessor)
- Lua 3.0 also allowed fallbacks to be tagged
- in Lua 5.0 this idea is implemented using *metatables* and *metamethods*

C API

- C API is what makes Lua embeddable language
- problem: how to exchange values between C and Lua (static vs. dynamic typing)
- solution in the latest versions: *abstract stack* that the programmer uses to communicate variables between

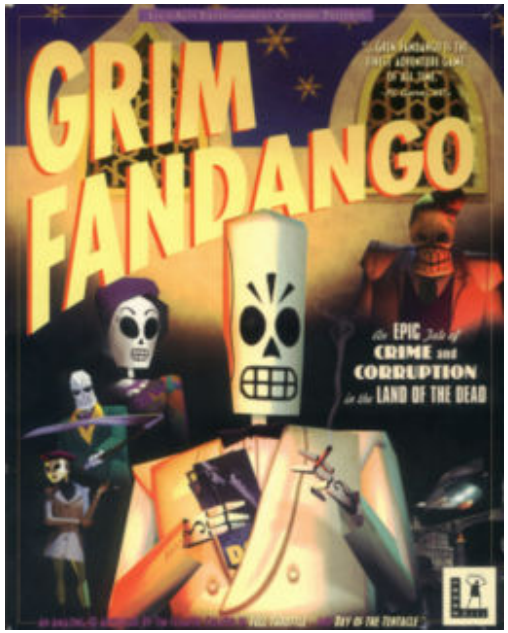
Improvements in the core

- C API in 4.0 has been extended and the standard libs are implemented on top of it
  - no longer any built-in functions
- since 5.0 the virtual machine is register-based (previously stack-based)

Conclusion

- Lua has shown a successful model of language growing
- not entirely the same as other open-source projects
- the authors avoided evolutionary traps
- Lua is popular in the industry

Where can you find it



- in games: Grim Fandango, WoW, The Sims and many others
- in companies: Adobe, Intel, Microsoft, etc.

Ales's Comments

- Good: good reading
- Bad: talking about minor syntax decisions or even version numbering
- Good: rasing the language
- Good: makes me want to use Lua

Thank you!

Your turn now.