February 6, 2002

Building a Multi-Language Interpreter Engine

Dan Sugalski
Or......
All your Interpreter are Belong to Us!
Our Languages of Interest

• Python
• Perl
• Ruby
• Scheme
• Tcl
• Objective C (A little)
What interpreters, especially for dynamic languages, provide general interpreter properties?
Resource Management

- Proper detection and destruction of dead objects
- OS resource management (threads, files, signals, and such like things)
- Memory collection and management
- Resource Management
OS Independence

• The whole world isn’t uniform
• Though they can still be worried about
  worry about platform-specific details
• Frees the programmer from having to
  feature not easily available
• Allow transparent emulation of
• OS
• Provides an abstract interface to the
• The whole world isn’t uniform
Rich type systems

• Interpreter's job to make complex data behave like simple data
• Makes non-traditional types easier for the programmer to use
• Requires a lot of work under the hood
• Easy extensibility here

PythonCon 10
Dynamic behaviour changes
• Dynamic recompilation
• Dynamic type behavior changes
• Makes classic optimizations somewhat difficult
- Matrix operations
- Argument generation
- Runtime class and method
- Curried functions
- Continuations
- Closures

High-level programming concept support
Safe Execution

Restrictive
- Static checking
- External access restrictions
- Paranoid runtime control flow
- Resource quotas

Safe Execution
Accommodating Specificity

Everyone does the same things differently, more or less
Object Models

- Mildly different
- Single/Multiple inheritance
- Object hierarchies differ
- Per-object variables and methods
Standard Libraries

- Every language has its own
- No two are exactly alike
- Only really an issue with functions provided by C routines
Syntax

- Significant differences between languages
- Generally just a parser issue
- Most significant issue for the programmer
- Least significant (almost) issue for the interpreter
Extensions

- Extension interfaces run from horrid (Perl) to very nice (Ruby)
  - Usually tied tightly to the implementation of the interpreter
  - Generally not considered part of the language
The easiest of the issues.

Ultimately a matter of speed more than anything else.

Languages of a class are trivial.

Semantic differences between most.

The easiest of the issues.

Semantics
How Parrot does all this stuff
Parrot's design goals

- Multi-language capable
- Longevity of the core design
- Portability
- Run Perl code fast
- Clean up all the gritty bits
- A good base for Perl's language features
- Longevity of the Perl language
We assume modern hardware. Lots of RAM handy.

A reasonable number of CPU registers.

Unpredictable branches expensive.

Main memory access expensive.

Good-sized L1 and L2 caches.

We assume modern hardware.
Parrot's a register machine

- Cache for the register-phobic
- Can be treated as a large named temp
- Avoids a lot of the common stack twiddling time-wasters
- Translates well to modern hardware
- Reduces by-name lookups of variables
- Reduces memory load/store

Parrot's a register machine
Simple types are basically built-in.

• Native int, native float, string, and PMCs
• PMCs are the "everything else" class
• Supports arbitrary-precision numbers
• Interface abstract to make adding new
  shortcuts for the optimizer easy

Native int, native float, string, and complex types
Split DOD & GC

- We check for dead objects and collect memory in separate phases
- Memory tends to get chewed up faster than objects die
- Most objects don’t need to do anything when they die
Easy extendability and embeddability

- Stable binary API
- Clean interface for extenders
- Simple and small interface for embedders
- Embedders have control over the interpreter’s environment (IO, ENV, command line args)
- Embedders have control over the environment.
- Internal details hidden
Portable

- Perl 5 runs (or has run) in 70+
- Platforms
- Linux, Win9x, Win32
- Every platform has something broken
- Not shooting for a lowest-common denominator
- Not about it
- Support for many Unixes, Win32, VMS, and Mac
Finally dump C’s stdio.

All I/O can be run through filters.

Appropriate byte, line, and record access.

Bulk read support everywhere.

Async I/O everywhere.

High-level I/O model.
Language-specific features are generally abstract.

- We don't mandate variable types or generic fallbacks are provided.
- Let's us punt on parts of the design and behaviours put things off for later.

Generally abstract.
Sort of OO under the hood

- Used as an abstraction layer
- OO support semi-abstract
- Neither are any CPUs to speak of
- All CPUs are not OO
- Appropriate

OO (of sorts, it's still all C)
Cross-language with Parrot
How to actually make it work
Variables

• Variable types know how to do things.
• Variables can be loaded on the fly.
• Operator overloading is generally implemented via variable vtable function.

Variables
Opcode libraries may also be loaded dynamically. Languages may define their own oplibs. Allows maximum performance for language-specific code with interpreter flexibility.
is easy

If we can manage Perl, everything else

Yacc to work with

Should be rather easier than lex &

to draw on

Has the full power of the Parrot engine

May be overridden lexically

Parser is general-purpose

Plugable parser
Inter-language calling

- We don’t, and can’t, guarantee 100% seamlessness.
- We don’t, and can’t, guarantee object hierarchies.
- Don’t guarantee object hierarchies.
- Provide a thunking layer for automatic type translations.
- Make things work at least as well as calling unspecialized C extensions.
- Usually better calling unspecialized C extensions.

Conventions
Questions?