PERL 6 and PARROT

An Overview

Presentation for NYSA

The New York System Administrators Association

June 9th, 2004

By Josh Rabinowitz

http://skateboarddirectory.com

Why Perl 6?

- Perl 5 codebase difficult to maintain
- Keep people from getting bored and doing other things
  (Jon Orwant's rumored rant)
- A paradigm shift - move Perl into the 21st century with features from other dynamic languages

Why Parrot?

- Concept started as an April Fool's joke by Simon Cozens on 4/1/2001, a mock interview with Larry Wall and Guido van Rossum 'detailing plans to merge Python and Perl into a new language called Parrot'.

- A few months later, when the Perl 6 internals began to become an independent part of the larger Perl 6 project, it was dubbed "Parrot".

Why Parrot? (Part II)

- Parrot is to be a ‘language-neutral runtime environment’ supporting all the features of dynamic languages such as Python, Ruby, Scheme, Befunge, BASIC, Perl 6, and others.
- Since Parrot is an independent project, it can continue more or less independently
- Parrot allows for easier and more consistent development of dynamic languages.
- Also allow for programs written in one language to call code and access data from in another (!)

Perl 6 Design Philosophy: Linguistic Considerations

- The Waterbed Theory of Complexity
- The Principle of Simplicity ("Perl 6 Grammers aren't simple. But they are complex at the language level in a way that allows simple solutions at the user level" p.17)
- The Principle of Adaptability ("Only dead languages stop changing")
- The Principle of Prominence: which words or parts of a line, or text appear most prominently
- The Principle of Context: Harder for the compiler, but can make programmatic structures more like natural language.

Perl 6 Design Philosophy: Linguistic Considerations (part II)

- The Principle of DWIM: This is cultural and largely linguistic.
- The Principle of Reuse: Use similar structures for similar purposes.
- The Principle of Distinction: Make different things look different. "Distinction and Reuse are in constant tension."
- Language cannot be seperated from culture.
- The Principle of Freedom. TMTOWTDI! "Programmers should have the freedom to choose how to express their code."
- The Principle of Borrowing: Don't be different just to be different, but use perl-ish structures to express borrowed concepts.

Perl 6 Design Philosophy: Architectural Considerations

- Perl should stay Perl
  - Keep easy things easy and hard things possible
- Be familiar to perl users
- Be mechanically translatable from perl 5
- Add Important New Features
  - Exceptions, delegation, multi-method dispatch, continuations, co-routines, and currying (to name a few). Many of these features are “traditionally difficult to understand”.
- Long Term Usability: Intended to last 20 years or more.

Major Perl 6 Features

- First: Any of this could be wrong or change.
- Good old perl variables: Scalars, Arrays, Hashes, References.
    - But references and dereferencing mostly happens for free
- New hash ‘methods’: %hash.keys, %hash.kv, %hash.values
- More contexts. Perl 5 had ‘list context’ and ‘scalar context’. Perl 6 has Scalar, List, and Hashlist context. In addition to Void, Boolean, Numeric, Integer, String, Object, List, non-flattening list, Flattening list, Hashlist, and Lazy List contexts.

More Major Perl 6 Features

- Properties and Traits. Traits are properties attached at compile time; properties can be changed at runtime. Can be attached to functions, methods, classes, grammers, rules, and ‘in parameter’ lists.
- Optional Type system. “my Array of Hash of Array of Int %hash”
- Whole slew of new operators: traditional arithmetic, string, bitwise, and conditional operators (many with a new syntax!)
- Also Vector operators such as >>+<<, Junction operators (all, any, one, none) [<< is the left-pointing guillemet]

Even More Major Perl 6 Features

- Less changes in basic control structures. Switch statement, ‘loop’ loop (was foreach)
- Exceptions: are classes that inherit from the Exception class. Thrown by throw, die, and fail (under use fatal). CATCH block ‘catches’ exceptions.
- Formal (named) parameters: sub func ($a, $b) {}
- Multimethods: chooses method based on signature list.
- Currying: uses ‘assuming’ method to create shortcuts to functions with preset parameter values.

Yet Even More Major Perl 6 Features

- Attributes are typed member variables of an object (uses ‘has’ keyword)
- Methods are always invoked on an object or class
- Inheritance uses ‘is’ keyword
- Rules (Perl regexes are now ‘so far beyond the formal defnition of regular expressions that we decided it was time for a more meaningful name’)
- Grammars: a collection of rules, typically for complex text parsing. Rules live in a grammer like methods live in a class.

Parrot Overview

- Parrot Architecture: Parser -> Compiler -> Optimizer -> Interpreter
  - Must be as fast as possible, as it imposes an upper limit on programs running on it.
  - Use abstraction to hide complexity
  - Must be stable both in terms of run-time robustness and also in terms of maintaining a stable interface for embedding.
- Parrot uses a bytecode and its own bytecode loader, which has the (theoretical) ability to load foreign bytecode formats and transform them into parrot-executable (byte)code.

About the Parrot Interpreter (Virtual Machine)

- Only designed for dynamic languages
- Uses a ‘register rich CISC CPU’ core design
- “The register-based architecture goes against the current trends in virtual machines, which favor stack-based approaches.” But lends “decades of compiler research to draw on”.
- There are 32 of each of four types of registers: PMC, string, integer and floating point. There are also seven separate stacks, one for each type of register, one for integers (used heavily by regexes, [I mean Rules]), a control stack (for exceptions and the like), and a general-purpose stack (for variables).

Strings in the Parrot Interpreter

- “Text data is deceptively complex”.
- Parrot makes strings a fundamental data type. This way all Parrot-targeted languages will handle strings consistently.
- Parrot strings know about encodings (such as ASCII or UTF-8) their respective character sets, and languages (for sorting and case-folding issues).

PMCs in the Parrot Interpreter

- In Parrot ‘variables’ use the PMC, or ‘Parrot Magic Cookie’: a base variable type for all target languages.

- PMCs encompass an abstract variable and a base data type.

- Each PMC has a vtable of function pointers, one for each thing the variable can do (set, get add, subtract, store, load, etc)

I/O, Events, Signals, and Threads in the Parrot Interpreter

- Fully asynchronous I/O
- Events and signals use the same I/O model
- “We could have decided to treat GUI events, network I/O, and file I/O all seperately, but there are plenty of systems around that demonstrate what a bad idea that is.”
- Since PMCs are not atomic types, threading is more difficult to do correctly (there is more to lock). Parrot uses the age-old ‘advisory locking’ model.

Not mentioned about the Parrot Interpreter:

- Garbage collection
- Signature-based dispatching
- Continuations: “invoked as if you never left the spot where they were created”
- Coroutines: “A subroutine or method that can suspend itself partway through, then pick up later where it left off”. Often used for iterators and generators, and to fake threads on systems that don’t natively support them.

More Reading:

- Allison Randal, Dan Sugalski & Leopold Tötsch. *Perl 6 Essentials*. O'Reilly, California, 2003

- The Perl Foundation. *Perl Development: Perl 6*. http://dev.perl.org/perl6/.

- The Perl Foundation. *ParrotCode: Parrot*. http://www.parrotcode.org/.