Parrot Update 2007

Jonathan Worthington

YAPC::EU::2007

Statistically

1 Year

1000s Of Commits

8 Releases

4 Hackathons

4 Hackathons

(The fifth one is happening right here, right now, at YAPC::EU::2007!)

1 New Architect

1 New License

1 New Object Model

1 New Bytecode File Format

Ain't Statistics Boring?

Parrot Update 2007

87% of people agree!*

* I totally made that number up. Use it for important stuff.

People

Parrot Update 2007

People

- Chip Salzenberg switched from the role of architect to pumpking
- Allison Randal became Parrot architect
- We wanted to move to monthly releases
  - Delegated the release process to a group of six people who will do two releases each a year – one every six months
  - Keeps the load of one person
  - We've been doing monthly releases since

The New Object Model

What was wrong with the old one?

- No support for roles – a big feature of the Perl 6 object model
- No introspection (aka reflection)
- Unclear how languages should implement their OO semantics in an interoperable way
- Inheriting from PMCs (classes implemented in C with some extra syntax) didn't really work too well; multiple inheritance just wasn't possible

Roles

- A group of methods and attributes
- Can't be instantiated on its own
- When a class does a role, the methods and attributes from the role are added to that class
- Composition is flattening: if a class tries to do two roles that have a method of the same name, it's an error
- But there are ways to resolve these conflicts

Roles - Implementation

- Added a Role PMC, which you can add methods and attributes to
- Classes have the add_role vtable method, which requests that they compose the role into themselves
- There is a way to specify a list of methods and attributes to not compose from a role => primitive for conflict resolution (you can implement the Perl 6 in terms of it, but it's more general)

Introspection

- Being able to take a class and find out about it
  - What is it called?
  - What namespace does it belong to?
  - What classes does it inherit from?
  - What roles does it do?
  - What methods does it have?
  - What attributes does it have?

Introspection

- Added inspect opcode

```
# Create class named LolCat
$P0 = new 'Class'
$P0.name('LolCat')
# All the introspection data...
$P1 = inspect $P0 # $P1 is Hash of data
# ...or just one item of it.
$P2 = inspect $P0, 'name'
say $P1 # LolCat
```

- Under the hood, it just calls inspect and inspect_str vtable methods

Supporting many different languages

- Parrot isn't just for Perl 6
- Different languages have quite different ways to do object orientation
  - There is no "one true implementation" that fits all of them
- We still want interoperability between different object models
- Solution: define a common interface that object models must implement

Example

- Some languages may allow addition of attributes even after the class has been instantiated
- In other languages classes are immutable once instantiated
- But they all allow addition of attributes somehow
- add_attribute is part of the standard interface, but a class system is free to implement it however it wishes

PMCProxy

- What if you want to do introspection on a PMC?
- When you write a class in PIR, you have an instance of the Class PMC to describe it
- There was no alternative for PMCs
- Added a PMCProxy PMC to describe a PMC
- Yes, it can describe itself ☺
- Implements the same interface as the Class PMC => consistency++, easier code gen.

Inheriting From PMCs

- Now looks just like inheriting from a class
- Use get_class opcode to get the PMC's PMCProxy object

```
$P0 = get_class 'Hash'
```

- Then add it as a parent to the new class

```
$P1 = new 'Class'
add_parent $P1, $P0
```

- The PMCProxy object sits in the list of parents, just as a Class object would

Inheriting From PMCs – Messy Guts

- Under the hood, quite a bit going on
- PMCs store state in C structures, default high level classes store it in an array
- Need to instantiate the PMCs we are inheriting from and keep them around to provide state storage
- Added a pointer to the PMC data structure to the "real self" so that down-calls would dispatch to any overridden methods

Bytecode File Improvements

New Bytecode Header Format

- Magic number not endian dependent
- Separate the idea of bytecode file format version and Parrot version
  - So Parrot upgrade need not invalidate the bytecode
  - Allow for multiple competing Parrot implementations in the future
- Support for storing UUIDs (User Unique IDs)
- New header format is implemented

Bytecode Annotations

- Need to provide high level language line numbers and file names to produce meaningful errors
- Need to be able to store any other compile time data other languages need, for example all the $? variables in Perl 6
- Bytecode annotations allow any Parrot instruction to be annotated with any key/value pair
- Designed, but not yet implemented

Bytecode PMCs

- At the moment, there is no way to work with bytecode files from within a Parrot program
- A bunch of PMCs have been specified to allow creation and manipulation of bytecode files from PIR
- Once implemented, will simplify the internals (less memory management work to do - just let the garbage collector do it for us)

Languages

Perl 6

- Now passes all of the sanity tests! ☺
- Running a slightly cut down version of the Perl 6 test harness
- Script to import some of the tests from the Pugs repository - we pass some of those too
- Basic expressions, scalars, arrays, hashes, method calls, arity-based multisubs, quoted terms, ranges (non-lazy), try blocks, $!, regexes, binding, listops, if and unless statements, chained operators and more!

Other Languages With Activity This Year

- APL
- BASIC
- ECMAScript
- Forth
- LISP
- Lua
- ParTcl (Tcl implementation)

Other Languages With Activity This Year

- Plumhead (PHP implementation)
- Pynine (Python implementation)
- Pheme (Scheme implementation)
- WMLScript

Compiler Tools

Parrot Compiler Toolkit

- Parrot Compiler Toolkit is the new name for Parrot's suite of compiler tools
- PGE = Parrot Grammar Engine
- TGE = Tree Grammar Engine
- PAST = Parrot Abstract Syntax Tree
- POST = Parrot Opcode Syntax Tree
- HLLCompiler = PMC that manages the compilation process and provides a standard interface

NQP

- NQP = Not Quite Perl
- Writing tree transforms in PIR takes quite a bit of effort, and is often a lot of code
- Now we can write them in NQP, which gets compiled down to PIR
- It's somewhat like a very limited Perl 6

Odds And Ends

Other Things That Deserve A Mention

- Much work has been done implementing "seat belts" – things that help us avoid writing bad code
- Much work has been done on portability, thanks to a microgrant; a side-effect of this is we can now built Parrot on C++ compilers too
- Loads of leaks plugged, bugs fixed, tests added – and a few performance improvements too

Final Thoughts

Parrot Update 2007

Optimism!

- It's been a good year for Parrot
- Several key bits of design that were missing or under-specified are now in good shape
- New object model unblocks things, including some languages
- Code base is in much better shape thanks to a focus on coding standards, as well as automated testing of adherence
- Importantly, lots of people are having fun!

Thank You!

Questions?