How not to Design a Scripting Language

Paul Biggar

Department of Computer Science and Statistics Trinity College Dublin

StackOverflow London, 28th October, 2009

How not to Design a Scripting Language

Paul Biggar

Department of Computer Science and Statistics Trinity College Dublin

StackOverflow London, 28th October, 2009



About me

- means I just submitted
- 2. So that's what this talk is about

About me

- PhD candidate, Trinity College Dublin
- **Topic:** Compilers, optimizations, scripting languages.

- PhD candidate, Trinity College Dublin
- Topic: Compilers, optimizations, scripting languages.



About me

- means I just submitted
- 2. So that's what this talk is about
- 3. Not much PHP

- PhD candidate, Trinity College Dublin
- **Topic:** Compilers, optimizations, scripting languages.

PhD Dissertation

Design and Implementation of an Ahead-of-time PHP Compiler

phc (http://phpcompiler.org)

About me

- PhD candidate, Trinity College Dublin
- Topic: Compilers, optimizations, scripting languages.

PhD Dissertation

Design and Implementation of an Ahead-of-time PHP Compiler

phc (http://phpcompiler.org)



How not to design a scripting language

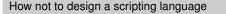
How not to design a scripting language

Compilers

Scripting Languages

- Compilers
- Scripting Languages





Compilers

- Scripting Languages
- Speed

How not to design a scripting language

- Compilers
- Scripting Languages
- Speed



 well, no-one actually has a good definition of scripting language!

2. I'm not talking about Bash or Powershell or VB, or some little language you wrote last week What is a scripting language?

Javascript

• Lua

Perl

• PHP

Python

Ruby

What is a scripting language?

- Javascript
- Lua
- Perl
- PHP
- Python
- Ruby



 well, no-one actually has a good definition of scripting language!
 I'm not talking about Bash or

about Bash or Powershell or VB, or some little language you wrote last week

3. exposing their internals

What is a scripting language?

- Javascript
- Lua
- Perl
- PHP
- Python
- Ruby

Common Features:

- Dynamic typing
- Duck typing
- · Interpreted by default
- FFI via C API

What is a scripting language?

- Javascript
- Lua
- Perl
- PHP
- Python
- Ruby

Common Features:

- Dynamic typing
- Duck typing
- Interpreted by default
- FFI via C API



How not to Design a Scripting Language

Language implementation

- 1. Reads one line at a time (kinda)
- 2. hence used in many scripting langs

• Interpreters: Easy, portable

Language implementation

• Interpreters: Easy, portable



- 1. Converts source code into machine code programs (kinda)
- 2. Lots of time to optimize

- Interpreters: Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*

Language implementation

- Interpreters: Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*



- 1. 2 months, Joel, Dragon
- 2. For intermediate and advanced, there are also much better books

- Interpreters: Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*

NOT THE DRAGON BOOK

Engineering a Compiler by Cooper/Torczon

Modern Compiler Implementation in X by Appel

Language implementation

- Interpreters: Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*

NOT THE DRAGON BOOK

Engineering a Compiler by Cooper/Torczon

Modern Compiler Implementation in X by Appel



- 1. Amazing optimizations; Hotspot; dispatch, exceptions and arithmetic
- 2. Shockingly difficult to write

- Interpreters: Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*
- Just-in-time compilers: Very difficult, unportable, fast interpreter.

Language implementation

- Interpreters: Easy, portable
- **Compilers:** Not too hard, sometimes portable, *optimizations*
- Just-in-time compilers: Very difficult, unportable, *fast interpreter*.



What's right with scripting languages?

- 1. dont dwell here l'm not here to convince you
- 2. the bad things dont touch these



- 1. dont dwell here l'm not here to convince you
- 2. the bad things dont touch these
- 3. Python, Ruby; JS too in a certain way

Elegant and well designed,

What's right with scripting languages?

• Elegant and well designed,



- 1. dont dwell here l'm not here to convince you
- 2. the bad things dont touch these
- 3. get things done, as Joel would say

Elegant and well designed,
 High level of abstraction,

What's right with scripting languages?

- Elegant and well designed,
- High level of abstraction,



- 1. dont dwell here l'm not here to convince you
- 2. the bad things dont touch these
- avoids many problems inherent in Java, C# and C++: verbosity, type systems

- Elegant and well designed,
- High level of abstraction,
- Oynamic typing (and duck typing).

What's right with scripting languages?

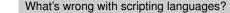
- Elegant and well designed,
- High level of abstraction,
- **③** Dynamic typing (and duck typing).

Symptoms: Speed, Portability

 portable to .Net, JVM, etc?
 not fast means you cant use it as much What's wrong with scripting languages?

Symptoms: Speed, Portability





Symptoms: Speed, Portability

- 1. portable to .Net, JVM, etc?
- 2. not fast means you cant use it as much

Problem: Language designed for interpreters

Run-time source code execution

What's wrong with scripting languages?

Symptoms: Speed, Portability

Problem: Language designed for interpreters

• Run-time source code execution



Symptoms: Speed, Portability

- 1. portable to .Net, JVM, etc?
- 2. not fast means you cant use it as much

Problem: Language designed for one specific interpreter

- Run-time source code execution
- Foreign Function Interface

What's wrong with scripting languages?

Symptoms: Speed, Portability

Problem: Language designed for one specific interpreter

- Run-time source code execution
- Foreign Function Interface



- 1. define FFI
- 2. glue code in C; wrap things in python data structures; expose interpreter internals
- 3. Works for all except JS - I'm using Python as example

FFI

FFI

Foreign Function Interface based on CPython interpreter

- Access to C libraries
- Script C applications using Python scripts
- Rewrite hot code in C

Foreign Function Interface based on CPython interpreter

- Access to C libraries
- Script C applications using Python scripts
- Rewrite hot code in C

FFI (good) implications

Libraries not that slow

• Can break out of Python for slow code.

FFI (good) implications

- Libraries not that slow
- Can break out of Python for slow code.



 nice things about python are lost (high level, elegant

- Language is allowed to be slow
- Must break out of Python for speed.

FFI (bad) implications

- Language is allowed to be slow
- Must break out of Python for speed.



FFI (worse) implications

Legacy issues

FFI (worse) implications

• Legacy issues



1. Jython,

IronPython, PyPy, cant use the same code

- 2. cant even reimplement own language
- 3. by constrast, look at JS

FFI (worse) implications

Legacy issues

Reimplementations

FFI

FFI (worse) implications

- Legacy issues
- Reimplementations



- 1. rule for life, really
- 2. like pyrex, ctypes, etc
- 3. no implementation specific code at all
- 4. import functions directly, and access them from within Python without a line of C
- 5. Ruby libFFI

Don't expose yourself!

 Importing functions into Python with a Domain Specific Language is good

FFI solution

Don't expose yourself!

 Importing functions into Python with a Domain Specific Language is good



- 1. rule for life, really
- 2. like pyrex, ctypes, etc
- 3. for reimplementation
- 4. no implementation specific code at all
- 5. import functions directly, and access them from within Python without a line of C
 6. Ruby libFFI

Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better

FFI solution

Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better



- 1. rule for life, really
- 2. like pyrex, ctypes, etc
- 3. no implementation specific code at all
- 4. import functions directly, and access them from within Python without a line of C
- 5. Ruby libFFI

Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best

FFI

FFI solution

Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best



- 1. rule for life, really
- 2. like pyrex, ctypes, etc
- 3. no implementation specific code at all
- 4. import functions directly, and access them from within Python without a line of C
- 5. Ruby libFFI

Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best
- Any reimplementation can reuse the same libraries without any modifications
- CPython itself can change without hassle

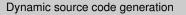
FFI

FFI solution

Don't expose yourself!

- Importing functions into Python with a Domain Specific Language is good
- Only one way of FFI is better
- Declarative is best
- Any reimplementation can reuse the same libraries without any modifications
- CPython itself can change without hassle





 Should be used for source inclusion. But because they're interpreted, they can be used for:

• eval and dynamic include/import

Compiled and interpreted models

Dynamic source code generation

• eval and dynamic include/import



 Should be used for source inclusion. But because they're interpreted, they can be used for:

- eval and dynamic include/import
 - meta-programming
 - eval (mysql_read (...)[0]);

Compiled and interpreted models

Dynamic source code generation

- eval and dynamic include/import
 - meta-programming
 - **eval** (mysql_read (...)[0]);



 Should be used for source inclusion. But because they're interpreted, they can be used for:

- \bullet eval and dynamic include/import
 - meta-programming
 - .rc files
 - username = "myname"
 password = "mypass"
 server = "srv.domain.com"

Compiled and interpreted models

Dynamic source code generation

- eval and dynamic include/import
 - meta-programming
 - .rc files

username = "myname"
password = "mypass"
server = "srv.domain.com"



 Should be used for source inclusion. But because they're interpreted, they can be used for:

- eval and dynamic include/import
 - meta-programming
 - .rc files
 - localization

\$lang = ...; include ("localisation/locale.\$lang.php");

Compiled and interpreted models

Dynamic source code generation

- eval and dynamic include/import
 - meta-programming
 - .rc files
 - localization

\$lang =;
include ("localisation/locale.\$lang.php");



Compiled and interpreted models

Dynamic source code generation

1. go to next slide straight away

We don't even know the full program source!!

We don't even know the full program source!!



- 1. self-perpetuating cycle
- 2. shame, scripting languages could really use static analysis
- high-level means high level optimizations? no
- 4. redundency elimination - some cant be done by hand

So they can't be compiled (ahead-of-time)

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

Compiled and interpreted models

So they can't be compiled (ahead-of-time)

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen



- 1. self-perpetuating cycle
- 2. shame, scripting languages could really use static analysis
- high-level means high level optimizations? no
- 4. redundency elimination - some cant be done by hand

So they can't be compiled (ahead-of-time)

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
for (i = 0; i < strlen(t); i++)
{
    s[i] = t[i];
}</pre>
```

Compiled and interpreted models

So they can't be compiled (ahead-of-time)

Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
for (i = 0; i < strlen(t); i++)
{
    s[i] = t[i];
}</pre>
```



- 1. self-perpetuating cycle
- 2. shame, scripting languages could really use static analysis
- high-level means high level optimizations? no
- 4. redundency elimination - some cant be done by hand

So they can't be compiled (ahead-of-time) Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
_temp = strlen(t);
for (i = 0; i < _temp; i++)
{
   s[i] = t[i];
```

Compiled and interpreted models

So they can't be compiled (ahead-of-time)

Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

```
t = ...;
_temp = strlen(t);
for (i = 0; i < _temp; i++)
{
    s[i] = t[i];
}</pre>
```



- 1. self-perpetuating cycle
- 2. shame, scripting languages could really use static analysis
- high-level means high level optimizations? no
- 4. redundency elimination - some cant be done by hand

So they can't be compiled (ahead-of-time) Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

alert (\$('li').get(0).nodeName);

Compiled and interpreted models

So they can't be compiled (ahead-of-time)

Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

alert (\$('li').get(0).nodeName);



- 1. self-perpetuating cycle
- 2. shame, scripting languages could really use static analysis
- high-level means high level optimizations? no
- 4. redundency elimination - some cant be done by hand

So they can't be compiled (ahead-of-time) Downsides:

- Must use FFI for speed
- Static analysis
- Cool optimizations can't happen

alert (\$('li')[0].nodeName);

Compiled and interpreted models

So they can't be compiled (ahead-of-time)

• Must use FFI for speed

- Static analysis
- Cool optimizations can't happen

alert (\$('li')[0].nodeName);



- instead, they must be JIT compiled. We'll still never get cool optimizations
- 2. worse, the research just gets us, at run-time, expensively, information we could get at compile-time.

JIT compiled

Tracemonkey

http://hacks.mozilla.org/2009/07/tracemonkey-overview/

Compiled and interpreted models

JIT compiled

Tracemonkey

http://hacks.mozilla.org/2009/07/tracemonkey-overview/



- instead, they must be JIT compiled. We'll still never get cool optimizations
- 2. worse, the research just gets us, at run-time, expensively, information we could get at compile-time.

JIT compiled

Tracemonkey

http://hacks.mozilla.org/2009/07/tracemonkey-overview/

Type Analysis for Javascript

Simon Holm Jensen, Anders Møller and Peter Thiemann SAS '09 http://www.brics.dk/TAJS/

Compiled and interpreted models

JIT compiled

Tracemonkey

http://hacks.mozilla.org/2009/07/tracemonkey-overview/

Type Analysis for Javascript

Simon Holm Jensen, Anders Møller and Peter Thiemann SAS '09 http://www.brics.dk/TAJS/



Compiled and interpreted models

Fix at **language** design time

No dynamic include; no eval.
Compile-time meta-programming

Compile-time meta-prog

1. do it like Perl, or

C++

Fix at language design time

- No dynamic include; no eval.
 - Compile-time meta-programming



Compiled and interpreted models

Fix at language design time

Fix at **language** design time

- No dynamic include; no eval.
 - Compile-time meta-programming
 - .rc files

1. sandbox them

- No dynamic include; no eval.
 - Compile-time meta-programming
 - .rc files



Compiled and interpreted models

Fix at language design time

Fix at **language** design time

1. With a compiled model, we know all the files

- No dynamic include; no eval.
 - Compile-time meta-programming
 - .rc files
 - localization

- No dynamic include; no eval.
 - Compile-time meta-programming
 - .rc files
 - localization



Doing it right

unfortunately chose a paradigm that nobody knows: stack-based; whoops!

Factor

- compiled model
- compile-time meta-programming
- declarative FFI

Compiled and interpreted models

Doing it right

Factor

- compiled model
- compile-time meta-programming
- declarative FFI



- 1. this is the shit we should be researching, instead of finding ways around intracaible problems
- 2. javascript 10px
- 3. table based
- 4. should be trivial

Open research problems

- Optimizing boxing
- High-level optimizations
- Combining ahead-of-time and JIT compilation

Compiled and interpreted models

Open research problems

- Optimizing *boxing*
- High-level optimizations
- Combining ahead-of-time and JIT compilation



Conclusion

Compiled and interpreted models

Conclusion

Design the next scripting language right

Design the next scripting language right



How not to Design a Scripting Language