Language Composition
MMNet 2015

Edd Barrett
Lukas Diekmann
Laurence Tratt
Carl Friedrich Bolz

Software Development Team
2013-07-15
Roadmap

- Language composition background.

- Challenges.

- Our approach.

- Concrete Example: PHP + Python.
“The ability to write a computer program in a mix of programming languages.”
Why Compose Languages?

- Parts of a program are expressed best with different languages.
  - User Interface
  - String manipulation
  - Statistical analysis
  - Constraint solving
  - …

- Performance.
  - Start writing in expressive language X.
  - Port bottlenecks to fast language Y.

- Language migration.
  - Gradual reimplementation.
Most languages have a Foreign Function Interface
Most languages have a Foreign Function Interface

File1 (Lang A) \[\text{FFI call} \rightarrow\] File2 (Lang B)

Coarse grain
Most languages have a Foreign Function Interface

File1 (Lang A) ~ File2 (Lang B)

FFI call

Coarse grain

Limited choice of langs

(second lang is nearly always C)
Our Aims:

- Fine-grain composition
  - Composition in the same file
  - Mix \{methods, functions, expressions\}
  - Integrate scoping

- Arbitrary languages
  - For now, dynamic languages.

- Make the composition fast.
Breaking it Down

PL X

PL Y

PL Z
Breaking it Down

PL X
- Syntax
- Runtime

PL Y
- Syntax
- Runtime

PL Z
- Syntax
- Runtime
Composing Syntax

\[
\text{PL X} \quad \begin{aligned}
\text{grammar} &::= \ldots \\
\text{expr} &::= \ldots \\
\text{term} &::= \ldots \\
\text{func} &::= \ldots
\end{aligned} \\
\text{PL Y} \quad \begin{aligned}
\text{grammar} &::= \ldots \\
\text{expr} &::= \ldots \\
\text{term} &::= \ldots \\
\text{func} &::= \ldots
\end{aligned} \\
\text{PL Z} \quad \begin{aligned}
\text{grammar} &::= \ldots \\
\text{expr} &::= \ldots \\
\text{term} &::= \ldots \\
\text{func} &::= \ldots
\end{aligned}
\]

\[\bigcup\]

Easy?
Composing Syntax

- LR → Possibly undefined.
- PEG → Shadows.
- GLR → Ambiguous.
Syntax Directed Editing

```java
public class Say extends <none> implements <none> {

    private String textchanged;
    <<properties>>
    <<initializer>>
    public Say(String text) {
        <<no statements>>
    }

    <<methods>>

    <<nested classifiers>>
}
```

Poor editing experience.
Composing Runtimes

RT X

RT Y

RT Z

Easy?
Runtime composition

PL X

Interpreter

PL Y

Interpreter

C/C++
Runtime composition

Too slow
Runtime composition

Too much engineering
Runtime composition

PL X
Interpreter

JIT Compiler

PL Y
Interpreter

JVM/CLR

12/39 HTTP://SOFT-DEV.ORG/
Runtime composition

Semantic mismatch
Summary:

We need a practical way of composing syntax and runtimes.
Our Approach

Summary:

We need a practical way of composing syntax and runtimes.

↓

Language Boxes + Meta-tracing
Language Boxes

- Borrows ideas from SDE.
- Palatable editing experience.
- Simple and practical way to compose grammars.
Begin writing Java code

```java
for (string s :
```
Language Boxes: E.g. Java + SQL

for (string s :
Open SQL language box
for (string s : SELECT * FROM tbl WHERE

Write SQL code

Language Boxes: E.g. Java + SQL
for (string s : 
SELECT * FROM tbl WHERE name = this.name;) {
}
How Does this Apply to VM Composition?
How Does this Apply to VM Composition?

PL X
Interpreters

PL Y

Glue

Metatracing

PL Z
Interpreter
Tracing JIT
Summarising our Approach

- Editing with Language boxes.
  - Traditional “code editor” look and feel.
  - Practical syntactic composition.

- Interpreter Composition with Meta-tracing
  - Relatively little engineering effort.
  - Compose any two languages written in RPython.
  - Language agnostic JIT optimisations.
Our Compositions
Our Language Compositions

Eco + RPython
Our Language Compositions

- Python + Prolog
- Python + PHP
- Python + SQLite
Our Language Compositions

- Python + Prolog
- Python + PHP = PyHyp
- Python + SQLite
PyHyp
PyHyp

Software Development Team

http://soft-dev.org/
Features of PyHyp

- **FFI-like features**
  - Calling Python functions and methods from PHP
  - Calling PHP functions and methods from Python
  - Automatic type “conversion”

- **Advanced features**
  - Adds support for references to Python
  - Arbitrary nesting of foreign functions
  - Cross-language scoping
  - Python expressions in PHP
  - “Embedding” Python methods inside PHP classes
  - Access modifiers
Edit/Execute

PHP + Python Code → Intermediate (x-lang interfaces) → PyHyp VM

Eco

PyHyp VM

25 / 39

http://soft-dev.org/
Implementing desired behaviour: relatively easy

Deciding the correct behaviours: hard

“Semantic friction”

Compromises sometimes must be made.
Example: Collection types across languages.
Semantic Friction: Array/Dict/List Conversions

Language Threshold

PHP

int
str
obj

Python

int
str
"adapted"
Semantic Friction: Array/Dict/List Conversions

Language Threshold

PHP

- array
  - list
  - integer keys

- array
  - dict
  - mixed keys

Python

- list
- dict
Semantic Friction: Array/Dict/List Conversions

PHP ← Language Threshold → Python

array ← question mark → dict

list
Semantic Friction: Array/Dict/List Conversions

PHP

Language Threshold

Python

- **array** (int keys)
- **array** (mixed keys)
- **list** (array)
- **dict** (array)

---

Software Development Team

28 / 39

http://soft-dev.org/
Semantic Friction: Array/Dict/List Conversions

PHP ⇄ Language Threshold ⇄ Python

$a = \text{array}

int keys

$a["x"] = 4

array

mixed keys

list array
Semantic Friction: Array/Dict/List Conversions

Language Threshold

PHP → Language Threshold → Python

$a = \text{array}_{\text{int keys}}$

$a["x"] = 4$

array

mixed keys

list

array

Inconsistent list!
Semantic Friction: Array/Dict/List Conversions

PHP  Language Threshold  Python

array  as_list()  list

array  dict

array

29/39 HTTP://SOFT-DEV.ORG/
Experimental Evaluation
Benchmarks

Microbenchmarks

“Larger” benchmarks

Benchmark Variants
Benchmarks

Variant 1
PHP

Variant 3
PHP + Python

Variant 2
Python

Variant 4
Python + PHP
Composed variant on PyHyp should perform “close” to mono variants on constituent interpreters.

Aim for between 1-2x slower. 3x is too slow.

For completeness, benchmark against other PHP and Python implementations too.
## Microbenchmarks: Relative to PyHyp Variant3

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CPython</th>
<th>HHVM</th>
<th>HippyVM</th>
<th>PyHyp&lt;sub&gt;m&lt;/sub&gt;</th>
<th>PyPy</th>
<th>Zend</th>
</tr>
</thead>
<tbody>
<tr>
<td>instchain</td>
<td>22.172</td>
<td>6.209</td>
<td>0.969</td>
<td>0.967</td>
<td>0.477</td>
<td>24.248</td>
</tr>
<tr>
<td></td>
<td>±0.0859</td>
<td>±0.0234</td>
<td>±0.0036</td>
<td>±0.0039</td>
<td>±0.0019</td>
<td>±0.1191</td>
</tr>
<tr>
<td>l1a0r</td>
<td>71.633</td>
<td>3.770</td>
<td>1.230</td>
<td>1.230</td>
<td>1.231</td>
<td>37.752</td>
</tr>
<tr>
<td></td>
<td>±1.4869</td>
<td>±0.0793</td>
<td>±0.0255</td>
<td>±0.0256</td>
<td>±0.0298</td>
<td>±1.1719</td>
</tr>
<tr>
<td>l1a1r</td>
<td>76.171</td>
<td>3.000</td>
<td>1.285</td>
<td>1.285</td>
<td>1.144</td>
<td>41.052</td>
</tr>
<tr>
<td></td>
<td>±0.1207</td>
<td>±0.0038</td>
<td>±0.0003</td>
<td>±0.002</td>
<td>±0.0077</td>
<td>±0.3498</td>
</tr>
<tr>
<td>lists</td>
<td>7.485</td>
<td>0.879</td>
<td>0.966</td>
<td>0.977</td>
<td>0.520</td>
<td>16.106</td>
</tr>
<tr>
<td></td>
<td>±0.0227</td>
<td>±0.0072</td>
<td>±0.0037</td>
<td>±0.0041</td>
<td>±0.0018</td>
<td>±0.0736</td>
</tr>
<tr>
<td>ref_swap</td>
<td></td>
<td>6.911</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
<td>55.360</td>
</tr>
<tr>
<td></td>
<td></td>
<td>±0.0054</td>
<td>±0.0003</td>
<td>±0.0003</td>
<td></td>
<td>±0.7395</td>
</tr>
<tr>
<td>return_simple</td>
<td>108.576</td>
<td>6.915</td>
<td>1.000</td>
<td>1.000</td>
<td>0.889</td>
<td>83.708</td>
</tr>
<tr>
<td></td>
<td>±0.2690</td>
<td>±0.0009</td>
<td>±0.0001</td>
<td>±0.0002</td>
<td>±0.0002</td>
<td>±0.7264</td>
</tr>
<tr>
<td>scopes</td>
<td>123.284</td>
<td>14.969</td>
<td>4.528</td>
<td>4.512</td>
<td>1.000</td>
<td>156.443</td>
</tr>
<tr>
<td></td>
<td>±1.5081</td>
<td>±0.0391</td>
<td>±0.0099</td>
<td>±0.0588</td>
<td>±0.0003</td>
<td>±0.5742</td>
</tr>
<tr>
<td>smallfunc</td>
<td>184.778</td>
<td>12.818</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>243.318</td>
</tr>
<tr>
<td></td>
<td>±0.3071</td>
<td>±0.0099</td>
<td>±0.0003</td>
<td>±0.0003</td>
<td>±0.0003</td>
<td>±0.8453</td>
</tr>
<tr>
<td>sum</td>
<td>299.582</td>
<td>19.083</td>
<td>1.000</td>
<td>1.000</td>
<td>0.874</td>
<td>427.513</td>
</tr>
<tr>
<td></td>
<td>±0.3659</td>
<td>±0.0172</td>
<td>±0.0005</td>
<td>±0.0005</td>
<td>±0.0003</td>
<td>±2.6441</td>
</tr>
<tr>
<td>sum_meth</td>
<td>328.894</td>
<td>23.714</td>
<td>0.998</td>
<td>0.999</td>
<td>0.873</td>
<td>456.739</td>
</tr>
<tr>
<td></td>
<td>±1.2870</td>
<td>±0.0955</td>
<td>±0.0030</td>
<td>±0.0030</td>
<td>±0.0026</td>
<td>±2.9270</td>
</tr>
<tr>
<td>sum_meth_attr</td>
<td>127.800</td>
<td>17.819</td>
<td>1.001</td>
<td>1.116</td>
<td>0.925</td>
<td>148.167</td>
</tr>
<tr>
<td></td>
<td>±0.1907</td>
<td>±0.2655</td>
<td>±0.0019</td>
<td>±0.0015</td>
<td>±0.0016</td>
<td>±0.6554</td>
</tr>
<tr>
<td>total_list</td>
<td>14.266</td>
<td>2.080</td>
<td>0.695</td>
<td>0.696</td>
<td>0.510</td>
<td>30.356</td>
</tr>
<tr>
<td></td>
<td>±0.0248</td>
<td>±0.0031</td>
<td>±0.0005</td>
<td>±0.0019</td>
<td>±0.0014</td>
<td>±0.1679</td>
</tr>
<tr>
<td>walk_list</td>
<td>4.869</td>
<td>0.373</td>
<td>0.773</td>
<td>0.774</td>
<td>1.099</td>
<td>10.700</td>
</tr>
<tr>
<td></td>
<td>±0.0340</td>
<td>±0.0025</td>
<td>±0.0060</td>
<td>±0.0107</td>
<td>±0.0082</td>
<td>±0.0846</td>
</tr>
<tr>
<td>Benchmark</td>
<td>CPython</td>
<td>HHVM</td>
<td>HippyVM</td>
<td>PyHyp_m</td>
<td>PyPy</td>
<td>Zend</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>deltetable</td>
<td>19.199</td>
<td>860.108</td>
<td>4.739</td>
<td>4.888</td>
<td>0.405</td>
<td>181.209</td>
</tr>
<tr>
<td></td>
<td>±0.6900</td>
<td>±31.1392</td>
<td>±0.1684</td>
<td>±0.1766</td>
<td>±0.0151</td>
<td>±6.7871</td>
</tr>
<tr>
<td>fannkuch</td>
<td>18.616</td>
<td>3.212</td>
<td>1.869</td>
<td>1.879</td>
<td>1.009</td>
<td>14.998</td>
</tr>
<tr>
<td></td>
<td>±0.0362</td>
<td>±0.0133</td>
<td>±0.0034</td>
<td>±0.0024</td>
<td>±0.0046</td>
<td>±0.1032</td>
</tr>
<tr>
<td>mandel</td>
<td>0.883</td>
<td>1.013</td>
<td>1.003</td>
<td></td>
<td></td>
<td>8.290</td>
</tr>
<tr>
<td></td>
<td>±0.0006</td>
<td>±0.0089</td>
<td>±0.0011</td>
<td></td>
<td></td>
<td>±0.0623</td>
</tr>
<tr>
<td>richards</td>
<td>28.291</td>
<td>12.726</td>
<td>0.745</td>
<td>0.766</td>
<td>0.531</td>
<td>27.081</td>
</tr>
<tr>
<td></td>
<td>±0.1091</td>
<td>±0.1296</td>
<td>±0.0036</td>
<td>±0.0044</td>
<td>±0.0026</td>
<td>±0.1445</td>
</tr>
<tr>
<td>Benchmark</td>
<td>CPython</td>
<td>HHVM</td>
<td>HippyVM</td>
<td>PyHyp$_m$</td>
<td>PyPy</td>
<td>Zend</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
<td>--------</td>
<td>---------</td>
<td>-----------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Geometric Mean</td>
<td>48.575</td>
<td>6.698</td>
<td>1.206</td>
<td>1.218</td>
<td>0.785</td>
<td>56.521</td>
</tr>
<tr>
<td></td>
<td>±0.1493</td>
<td>±0.0188</td>
<td>±0.0032</td>
<td>±0.0035</td>
<td>±0.0024</td>
<td>±0.1833</td>
</tr>
</tbody>
</table>
Conclusions

- Language boxes:
  - Practical composition of PL syntax.
  - Decent editor experience.

- Meta-tracing:
  - Compositions with relatively little effort.
  - Overall good performance.

- Implementing x-lang behaviours is easy.
- Designing x-lang behaviours is hard.
  - Thanks to semantic friction.
Future Work

- Debugging
  - Proper backtrace information.
  - Cross-language debugger.

- Compositions with >2 languages involved.

- Statically typed languages.
Language Boxes + Meta-tracing