Run-Fail-Grow: Creating Tailored Object-Oriented Runtimes

Guille Polito
@guillep
https://guillep.github.io/

G. Polito, L. Fabresse, N. Bouraqadi, S. Ducasse, in Journal of Object Technology
A(gain) need for small and fast startup

- Micro Services
- Container architectures
- Small devices
Lets look at an application...
MainApp>>start
logger := StdoutLogger new.
logger log: 'Application has started'.
"do something"
logger log: 'Application has finished'.

StdoutLogger>>newLine
stdout newLine.

StdoutLogger>>log: aMessage
stdout nextPutAll: Time now printString.
stdout nextPutAll: aMessage.
stdout newLine.

RemoteLogger>>log: aMessage
| socket |
socket := self newSocket.
socket nextPutAll: Time now printString.
socket nextPutAll: aMessage.
socket newLine.

RemoteLogger>>newSocket
"...."
"creates an instance of socket given some configuration"
There is *extra* stuff!
MainApp>>start
logger := StdoutLogger new.
logger log: 'Application has started'.
"do something"
logger log: 'Application has finished'.

StdoutLogger>>newLine
stdout newLine.

StdoutLogger>>log: aMessage
stdout nextPutAll: Time now printString.
stdout nextPutAll: aMessage.
stdout newLine.

RemoteLogger>>log: aMessage
  | socket |
  socket := self newSocket.
  socket nextPutAll: Time now printString.
  socket nextPutAll: aMessage.
  socket newLine.

RemoteLogger>>newSocket
  "...."
  "creates an instance of socket given some configuration"
MainApp>>start
logger := StdoutLogger new.
logger log: 'Application has started'.
"do something"
logger log: 'Application has finished'.

StdoutLogger>>newLine
stdout newLine.

StdoutLogger>>log: aMessage
stdout nextPutAll: Time now printString.
stdout nextPutAll: aMessage.
stdout newLine.

RemoteLogger>>log: aMessage
| socket |
socket := self newSocket.
socket nextPutAll: Time now printString.
socket nextPutAll: aMessage.
socket newLine.

RemoteLogger>>newSocket
"...."
"creates an instance of socket given some configuration"
MainApp>>start
logger := StdoutLogger new.
logger log: 'Application has started'.
"do something"
logger log: 'Application has finished'.

StdoutLogger»newLine
stdout newLine.

StdoutLogger»log: aMessage
stdout nextPutAll: Time now printString.
stdout nextPutAll: aMessage.
stdout newLine.

RemoteLogger»log: aMessage
| socket |
socket := self newSocket.
socket nextPutAll: Time now printString.
socket nextPutAll: aMessage.
socket newLine.

RemoteLogger»newSocket
"...."
"creates an instance of socket given some configuration"
MainApp>>start
logger := StdoutLogger new.
logger log: 'Application has started'.
"do something"
logger log: 'Application has finished'.

StdoutLogger>>newLine
stdout newLine.

StdoutLogger>>log: aMessage
stdout nextPutAll: Time now printString.
stdout nextPutAll: aMessage.
stdout newLine.

RemoteLogger>>log: aMessage
| socket | 
| socket := self newSocket. 
| socket nextPutAll: Time now printString. 
| socket nextPutAll: aMessage. 
| socket newLine. 

RemoteLogger>>newSocket
"...."
"creates an instance of socket given some configuration"
But there is not only **BLOAT in application code**

- Core Libraries take place and are distributed with the language
- Runtime objects take place also

  Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. Visualizing objects and memory usage. In Smalltalks 2010

- And some of those are runtime meta-data, used for reflection
Research questions

Q: Can we tailor the runtime?

Q: How small can they be?
Our solution in a nutshell

a runtime virtualisation infrastructure

+ 

a run-fail-grow algorithm
Tornado Run-Fail-Grow

hypervisor

Virtualized Runtime

(Empty)
Tornado Run-Fail-Grow

1. run

hypervisor

Virtualized Runtime
Tornado Run-Fail-Grow

hypervisor

Virtualized Runtime

2. fail
Tornado Run-Fail-Grow

3. copy missing

hypervisor

Virtualized Runtime
Tornado Run-Fail-Grow
Tornado Run-Fail-Grow

1. run
2. fail
3. copy missing
4. resume

hypervisor

Virtualized Runtime
How small can we get?

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Size (KB)</th>
<th>%Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>11</td>
<td>99.99%</td>
</tr>
<tr>
<td>Reflective App</td>
<td>32</td>
<td>99.83%</td>
</tr>
<tr>
<td>Factorial 100 + I/O</td>
<td>89</td>
<td>99.39%</td>
</tr>
<tr>
<td>Seaside Counter</td>
<td>573</td>
<td>96.73%</td>
</tr>
</tbody>
</table>
Comparison with Traditional Tailoring Solutions

<table>
<thead>
<tr>
<th>criteria</th>
<th>Dedicated Platforms</th>
<th>Static/Hybrid Analysis</th>
<th>Dynamic Analysis</th>
<th>Tornado</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Comparison with Traditional Tailoring Solutions

<table>
<thead>
<tr>
<th><strong>criteria</strong></th>
<th>Dedicated Platforms</th>
<th>Static/Hybrid Analysis</th>
<th>Dynamic Analysis</th>
<th>Tornado</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base libraries</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Third-party libraries</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legacy Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reflection Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General Purpose Infrastructure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configurability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Typing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimality</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison with Traditional Tailoring Solutions

<table>
<thead>
<tr>
<th>criteria</th>
<th>Dedicated Platforms</th>
<th>Static/Hybrid Analysis</th>
<th>Dynamic Analysis</th>
<th>Tornado</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base libraries</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Third-party libraries</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Legacy Code</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Reflection Support</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>General Purpose Infrastructure</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Configurability</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Dynamic Typing</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Minimality</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Completeness</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
</tbody>
</table>
Run-Fail-Grow: Creating Tailored Object-Oriented Runtimes

- Run-fail-grow approach for tailoring
- Works on dynamic languages
- Small memory footprint results (10k!)

G. Polito, L. Fabresse, N. Bouraqadi, S. Ducasse, in Journal of Object Technology

Guille Polito
@guillep
https://guillep.github.io/