Efficient Layered Method Execution in ContextAmber

COP 2015

Matthias Springer, Jens Lincke, Robert Hirschfeld

Hasso Plattner Institute, Software Architecture Group

July 5, 2015
Introduction

- Implemented **ContextAmber**: layer-based COP library, written in Smalltalk, compiled to JavaScript
- Optimizations for ContextAmber: make *layered method execution faster*
- Running example: Vector Graphics Debugging
Which COP is it?

- Layer-based COP for class-based object-oriented programming
- Layer activation **globally** (+scoped) and **per object**
- **Explicit layer activation** only
  (i.e. no declarative layer activation or `activeLayer` method override)
Problem: Why is ContextAmber slow?

What happens when a layered method is invoked:

1. Compute which layers are active for the receiver

   | global:      | (L1, L2, L3, L4)         |
   | O1:          | (+L5, -L2, -L3)          |
   |             | = (L1, L4, L5)           |

   | O2:          | (-L1 + L1)               |
   |             | = (L2, L3, L4, L1)       |

2. Repeatedly do:

   2.1 Find next partial method
       L2       L3       L4       L1

   2.2 Dispatch to partial method
Solution

- **Cache active layers** on a per-object basis
- **Aggressive inlining**: remove all partial method dispatches
- **Inlined method caching**
What’s Next?

Biggest overhead: looking up and dispatching to next partial method
Partial Method Inlining

(layered method invocation)
→
compute layer comp. + signature
→ no
→ signature differs?
→ no
→ (execute method)

install method
→
generate method
→ partial method inlining
What’s Next?

Biggest overhead: **inlining methods**
every time the layer composition changes
Method Caches

(layered method invocation)

compute layer comp. + signature

no

signature differs?

yes

(composition already computed)

no

(execute method)

method in cache?

no

generate method + put in cache

yes

install method

partial method inlining

Hasso Plattner Institute, Software Architecture Group

Efficient Layered Method Execution

July 5, 2015
What’s Next?

Biggest overhead: calculating the current layer composition on every layered method execution
Layer Composition Changes

When does the layer composition change?

- Layer activated for an object
  → single object affected → dirty bit
- Layer activated globally
  → multiple objects affected → version number
Layer Composition Caching

(layered method invocation)

receiver dirty?

yes → composition signature caching

no →

receiver version out of date?

yes → compute layer comp. + signature

no →

signature differs?

yes → compute layer composition

no → (execute method)

method in cache?

no → generate method + put in cache

yes → install method

partial method inlining
What’s Next?

Biggest overhead: (probably) JIT trace invalidation every time a new layered method is installed
Instance-specific Method Inlining

Every **object** has its **own inlined method**.

- Layer composition change: nothing changed (different layer composition $\rightarrow$ different inlined method)
- Invoke `a.method` and `b.method`, and `a` and `b` have different layer compositions: no JIT trace invalidation anymore
What’s Next?

Performance is very close to performance without COP
### Benchmarks

Average, without first frame

![Bar chart showing runtime (ms) for different conditions: without ContextAmber, class-specific (cached), class-specific (uncached), instance-specific (cached), instance-specific (uncached). The conditions are: no layers, control point layer, control point layer (mixed). The values are as follows:

- No layers: 5.15, 5.32, 6.09, 5.41, 5.21
- Control point layer: 21.33, 21.67, 28.59, 21.48, 22.60
- Control point layer (mixed): 32.40, 16.29, 13.75]
Benchmarks

First frame only

- Without ContextAmber
- Class-specific (cached)
- Class-specific (uncached)
- Instance-specific (cached)
- Instance-specific (uncached)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Runtime (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No layers</td>
<td>7</td>
</tr>
<tr>
<td>Control point layer</td>
<td>126</td>
</tr>
<tr>
<td>Control point layer (mixed)</td>
<td>1937</td>
</tr>
<tr>
<td>N/A</td>
<td>246</td>
</tr>
<tr>
<td></td>
<td>236</td>
</tr>
<tr>
<td></td>
<td>1686</td>
</tr>
</tbody>
</table>
Future Work

- Methods are taken from a cache mapping composition signatures to inlined methods
- One method only is ever installed
- Next step: make method lookup aware of layer compositions
  - receiver type $\times$ composition signature $\rightarrow$ target method
  - Preserve JIT traces even if layer composition changes
Summary

(layered method invocation)

receiver dirty?
  yes
  composition signature caching
  no

receiver version out of date?
  yes
  compute layer comp. + signature
  no

signature differs?
  yes
  compute layer composition
  no

method in cache?
  yes
  generate method + put in cache
  no

install method

(layered method invocation)

Method Inlining

Method Caching

Layer Composition Caching