Metacircular Virtual Machine Layering for Run-Time Instrumentation

Erick Lavoie, Bruno Dufour, Marc Feeley
Université de Montréal
ericklavoie.com
Motivation
Context

- Program comprehension
- Benchmark generation
- Security invariant monitoring
- Hybrid analysis design
- Run-Time optimization design
Run-Time Instrumentation

- Object model operations
- Function calls
- Scope chain
- Control-flow
- ...

...
Direct Instrumentation of Virtual Machine

+ Guaranteed compliance
+ Integrated in a browser
+ Straightforward on a simple interpreter
- Needs maintenance
- Tied to a single browser
- Becoming more complex
Research Problem
Research Problem

How can we reify opaque aspects of JavaScript for high-level run-time instrumentation at a minimal cost in performance?
Approach
Metacircular Virtual Machine Layering

+ Differential implementation
+ Independent from Host VM implementation
- Run-time cost

JavaScript program

Metacircular JS VM

Host JS VM
Application
Program Comprehension

• What is the contribution of application data structures to its overall memory footprint?

• Where are these objects created with regard to the source code?

• What is the contribution of every function to the overall object creation?

• What is the abstract shape of those object in memory?
Run-Time Instrumentation

- Object model operations
- Function calls
- Scope chain
- Control-flow
- ...

12
Design Goals

• Sandboxing of application runtime environment

• Dynamically redefinable instrumentation, for activation and deactivation at run-time

• Specialization of reified operations, instrumented or not, at their call-site
Design

High-Performance Virtual Machine with a JIT Compiler

Source-to-Source JIT Compiler

- Function Calls, Object Ops
- Dynamically Redefinable Function Calls and Object Ops
- Message Sending

Object Representation

- Object Ops, Function Calls
- Litteral Object Creation

Instrumentation

- Instrumented Object Ops and Function Calls

Application

- Full JavaScript Support

Control-flow, Primitives, etc.

- Caching and Specialization

- Fast Global Function Calls
- Inline Caching and Specialization of Method Calls

Data Structures

- Containers
- Native Objects
Reifying Object Model Operations

```javascript
var o = {};
o.p = 42;
o.p;
delete o.p;
new F();
```

```javascript
var o = send(root.object, "__new__");
send(o, "__set__", "p", 42);
send(o, "__get__", "p");
send(o, "__delete__", "p");
send(F, "__ctor__");
```
Counting the Number of Property Accesses

```javascript
var getCounter = 0;

send(root.object, "__set__", "__get__", 
    function (name) {
        getCounter++; 
        return this.get(name);
    }
);
```
Reifying Function Calls

function (g) {g();};

h.call();

f();

o.p();

function (g) {
    send(g, "call");
    }

send(h, "call");

send(global, "f");

send(o, "p");
function send(obj, msg, ..args) {
    var method = obj.get(msg);
    return method.call(obj, ..args);
}
Semantics of Message Sending

function send(obj, msg, ..args) {
    var method = obj.get(msg);
    var callFn = method.get("call");
    return callFn.call(method, obj, ..args);
}
Intercepting Function Calls

function beforeCall() {
...
}

send(root.function, "__set__", "call", function (obj, ..args) {
    beforeCall(this, obj, ..args);
    return this.call(obj, ..args);
});
Optimization
Object Representation

- Encapsulate invariants of the implementation
- Provide fast object creation, accesses and updates
- Allow transparent per-object information collection
Basic Object Representation

Legend

- - - - prototype
- - - - proxied object
Special Object Representation

Legend
- - - > prototype
- - - - > proxied object
Object Representation

Operations

```javascript
var o = {};

o.p = 42;

o.p;

delete o.p;

root.object.create();

o.set("p", 42);

o.get("p");

o.delete("p");
```
Call-Site Specialization of Arguments Nb

```javascript
function f(g) {
    return g();
}
```

```javascript
new FunctionProxy(function f(g) {
    return (g instanceof FunctionProxy) ?
    g.proxiedObject.call($global) :
    error(g);
});
```
Call-Site Specialization of Arguments Nb

function f(g) {
    return g();
}

ew FunctionProxy(function f($this,g) {
    return (g instanceof FunctionProxy) ?
        g.proxiedObject($global) :
        error(g);
});
Call-Site Specialization of Arguments Nb

```javascript
function f(g) {
    return g();
}
```

```javascript
new FunctionProxy(function f($this, g) {
    return g.call($global);
});
```

```javascript
FunctionProxy.prototype.call =
function (obj) {
    return this.proxiedObject.apply(obj,
        Array.prototype.slice.call(arguments, 1));
};
```
Call-Site Specialization of Arguments Nb

```javascript
function f(g) {
    return g();
}
```

```javascript
new FunctionProxy(function f($this, g) {
    return g.callWith0Arg($global);
});
```

```javascript
FunctionProxy.prototype.callWith0Arg = function (obj) {
    return this.proxiedObject(obj);
};
```

```javascript
function invFn() { throw Error("Invalid Function"); }
```

```javascript
ObjectProxy.prototype.callWith0Arg = invFn;
```

```javascript
String.prototype.callWith0Arg = invFn;
```

...
Counting the Number of Property Accesses (revisited)

```javascript
var getCounter = 0;

send(root.object, "__set__/", "__get__/", 
    new FunctionProxy(
        function ($this, name) {
            getCounter++;
            return $this.get(name);
        }
    )
);
```
Empirical Evaluation
Baseline Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Pn</th>
<th>SM</th>
<th>V8</th>
<th>V8/Pn</th>
<th>SM/Pn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crypto</td>
<td>529.0</td>
<td>348.0</td>
<td>17025.0</td>
<td>32.2</td>
<td>0.7</td>
</tr>
<tr>
<td>DeltaBlue</td>
<td>82.8</td>
<td>249.0</td>
<td>19306.0</td>
<td>233.2</td>
<td>3.0</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>738.0</td>
<td>808.0</td>
<td>34170.0</td>
<td>46.3</td>
<td>1.1</td>
</tr>
<tr>
<td>NavierStokes</td>
<td>908.0</td>
<td>564.0</td>
<td>20947.0</td>
<td>23.1</td>
<td>0.6</td>
</tr>
<tr>
<td>RayTrace</td>
<td>156.0</td>
<td>560.0</td>
<td>19442.0</td>
<td>124.6</td>
<td>3.6</td>
</tr>
<tr>
<td>RegExp</td>
<td>441.0</td>
<td>781.0</td>
<td>3902.0</td>
<td>8.8</td>
<td>1.8</td>
</tr>
<tr>
<td>Richards</td>
<td>120.0</td>
<td>219.0</td>
<td>14149.0</td>
<td>117.9</td>
<td>1.8</td>
</tr>
<tr>
<td>Splay</td>
<td>118.0</td>
<td>1508.0</td>
<td>5850.0</td>
<td>49.6</td>
<td>12.8</td>
</tr>
<tr>
<td>V8 Score</td>
<td>270.0</td>
<td>524.0</td>
<td>14002.0</td>
<td>51.9</td>
<td>1.9</td>
</tr>
</tbody>
</table>
## Baseline Memory Usage

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Pn</th>
<th>V8</th>
<th>Pn/V8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crypto</td>
<td>56.0</td>
<td>20.0</td>
<td>2.8</td>
</tr>
<tr>
<td>DeltaBlue</td>
<td>33.0</td>
<td>20.0</td>
<td>1.6</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>128.0</td>
<td>20.0</td>
<td>6.4</td>
</tr>
<tr>
<td>NavierStokes</td>
<td>29.0</td>
<td>19.0</td>
<td>1.5</td>
</tr>
<tr>
<td>RayTrace</td>
<td>35.0</td>
<td>20.0</td>
<td>1.8</td>
</tr>
<tr>
<td>RegExp</td>
<td>54.0</td>
<td>22.0</td>
<td>2.5</td>
</tr>
<tr>
<td>Richards</td>
<td>28.0</td>
<td>18.0</td>
<td>1.6</td>
</tr>
<tr>
<td>Splay</td>
<td>84.0</td>
<td>97.0</td>
<td>0.9</td>
</tr>
</tbody>
</table>
## Instrumented Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Pn</th>
<th>Pn-spl</th>
<th>Pn-fast</th>
<th>Pn/Pn-spl</th>
<th>Pn/Pn-fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crypto</td>
<td>529.0</td>
<td>41.4</td>
<td>566.0</td>
<td>12.8</td>
<td>0.9</td>
</tr>
<tr>
<td>DeltaBlue</td>
<td>82.8</td>
<td>36.2</td>
<td>103.0</td>
<td>2.3</td>
<td>0.8</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>738.0</td>
<td>162.0</td>
<td>767.0</td>
<td>4.6</td>
<td>1.0</td>
</tr>
<tr>
<td>NavierStokes</td>
<td>908.0</td>
<td>51.4</td>
<td>871.0</td>
<td>17.7</td>
<td>1.0</td>
</tr>
<tr>
<td>RayTrace</td>
<td>156.0</td>
<td>85.1</td>
<td>158.0</td>
<td>1.8</td>
<td>1.0</td>
</tr>
<tr>
<td>RegExp</td>
<td>441.0</td>
<td>324.0</td>
<td>476.0</td>
<td>1.4</td>
<td>0.9</td>
</tr>
<tr>
<td>Richards</td>
<td>120.0</td>
<td>30.5</td>
<td>113.0</td>
<td>3.9</td>
<td>1.1</td>
</tr>
<tr>
<td>Splay</td>
<td>118.0</td>
<td>453.0</td>
<td>117.0</td>
<td>0.3</td>
<td>1.0</td>
</tr>
<tr>
<td>V8 Score</td>
<td>270.0</td>
<td>91.2</td>
<td>281.0</td>
<td>3.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

- Metacircular VM layering can be used to reify object model operations and function calls at a performance level 2x slower than a state-of-the-art interpreter
- Simple implementation (~1700 LOC runtime library excluding JS-to-JS translator)
Future Work

• Support DOM to integrate with a browser
• Apply approach to reify scope chain and control-flow for other applications
• Improve performance by exploiting dynamic recompilation to remove redundant checks
• Develop metacompilers to generate custom VMs for specific instrumentation tasks