MethodProxies: A Safe and Fast Message-Passing Control Library

Sebastian JORDAN MONTAÑO (1), Juan Pablo SANDOVAL ALCOCER (2), Guillermo POLITO (1), Stéphane DUCASSE (1), Pablo TESONE (1,3)

sebastian.jordan@inria.fr

1. Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL
2. Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile
3. Pharo Consortium








IWST ’24, July 2024, Lille, France

Objects communicate through messages

1. The message `#method:`
is sent to `aReceiver`

```
aReceiver method: args
```

Message-passing

2. Executes `#method:`
into the object
`aReceiver`

Pharo's VM

Message-passing control

The message is captured

1. The message `#method:` is sent to `aReceiver`

```
aReceiver method: args
```

2. A user-defined action is executed before the method's execution

#beforeAction

3. Executes `#method:` into the object `aReceiver`

Pharo's VM

4. An action is executed after the execution

#afterAction

Safe Message-Passing Control

- Meta-safe recursion
- Thread safety
- Safe handling of exceptions and non-local returns
- Uninstrumentation

Meta-safe recursion

The library should provide a **meta-safe recursion** prevention to manage recursions **originating from within the instrumented** code.

```
AClass>>foo
	handler before.
	"method code"
	handler after.

Handler>>before
	'foo method called' logMessage.
	anInstanceOfAClass foo.
```

Thread safety

It manage **meta-executions in a thread-specific** manner. It should ensure that meta-executions are marked **uniquely for each thread**.

Handling of exceptions and non-local returns

It must **ensure that the afterMethod executes** under all circumstances, whether an **exception** or a **non-local return** occurs.

```
AClass>>foo
	handler before.
	condition ifAbsent: [ ^ self ].
	handler after.
```

Uninstrumentation

It must **uninstrument all the methods** that were instrumented, restoring them to their original state.

Current Message-Passing Control Techniques

- Source code modification
- run:with:in method hook

Source Code Modification

```
"Before Instrumentation"
AClass>>foo
        | temp1 |
        temp1 := self doSomething.
        ^ temp1
```

Source Code Modification

```
"Before Instrumentation"
AClass>>foo
	| temp1 |
	temp1 := self doSomething.
	^ temp1
```

```
"After Instrumentation"
AClass>>foo
	self isMetaForActiveProcess ifFalse: [
	self runInMetaLevel: [ #beforeHandler ] ].
	[ | temp1 |
	temp1 := self doSomething.
	^ temp1 ] ensure: [
		self isMetaForActiveProcess ifFalse: [
			self runInMetaLevel: [ #afterAction ] ] ]
```

Source Code Modification

```
"Before Instrumentation"
AClass>>foo
        | temp1 |
        temp1 := self doSomething.
        ^ temp1
```

```
"After Instrumentation"
AClass>>foo
        self isMetaForActiveProcess ifFalse: [
        self runInMetaLevel: [ #beforeHandler ] ].
        [ | temp1 |
        temp1 := self doSomething.
        ^ temp1 ] ensure: [
            self isMetaForActiveProcess ifFalse: [
                self runInMetaLevel: [ #afterAction ] ] ]
```

Meta checking + before action

Source Code Modification

```
"Before Instrumentation"
AClass>>foo
        | temp1 |
        temp1 := self doSomething.
        ^ temp1
```

```
"After Instrumentation"
AClass>>foo
        self isMetaForActiveProcess ifFalse: [
        self runInMetaLevel: [ #beforeHandler ] ].
        [ | temp1 |
        temp1 := self doSomething.
        ^ temp1 ] ensure: [
                self isMetaForActiveProcess ifFalse: [
                        self runInMetaLevel: [ #afterAction ] ] ]
```

Original code

Source Code Modification

```
"Before Instrumentation"
AClass>>foo
        | temp1 |
        temp1 := self doSomething.
        ^ temp1
```

```
"After Instrumentation"
AClass>>foo
        self isMetaForActiveProcess ifFalse: [
        self runInMetaLevel: [ #beforeHandler ] ].
        [ | temp1 |
        temp1 := self doSomething.
        ^ temp1 ] ensure: [
            self isMetaForActiveProcess ifFalse: [
                self runInMetaLevel: [ #afterAction ] ] ]
```

Meta checking
+ After action

run:with:in

- During the lookup, if the VM does not find an instance of a CompiledMethod, it sends the message #run:with:in to the located method object.
- The run:with:in: technique replaces a compiled method instance with a ProxyObject understanding a run:with:in: message

run:with:in

```
ProxyObject >> run: selector with: args in: aReceiver
    | v |
    self isMetaForActiveProcess ifFalse: [
        self runInMetaLevel: [ #beforeHandler ] ].

    v := originalMethod valueWithReceiver: aReceiver arguments: args
    ] ensure: [
        self isMetaForActiveProcess ifFalse: [
            self runInMetaLevel: [ #afterHandler ] ] ]
    ^ v
```

Meta checking + before action

run:with:in

```
ProxyObject >> run: selector with: args in: aReceiver
    | v |
    self isMetaForActiveProcess ifFalse: [
        self runInMetaLevel: [ #beforeHandler ] ].

    v := originalMethod valueWithReceiver: aReceiver arguments: args
    ] ensure: [
        self isMetaForActiveProcess ifFalse: [
            self runInMetaLevel: [ #afterHandler ] ] ]
    ^ v
```

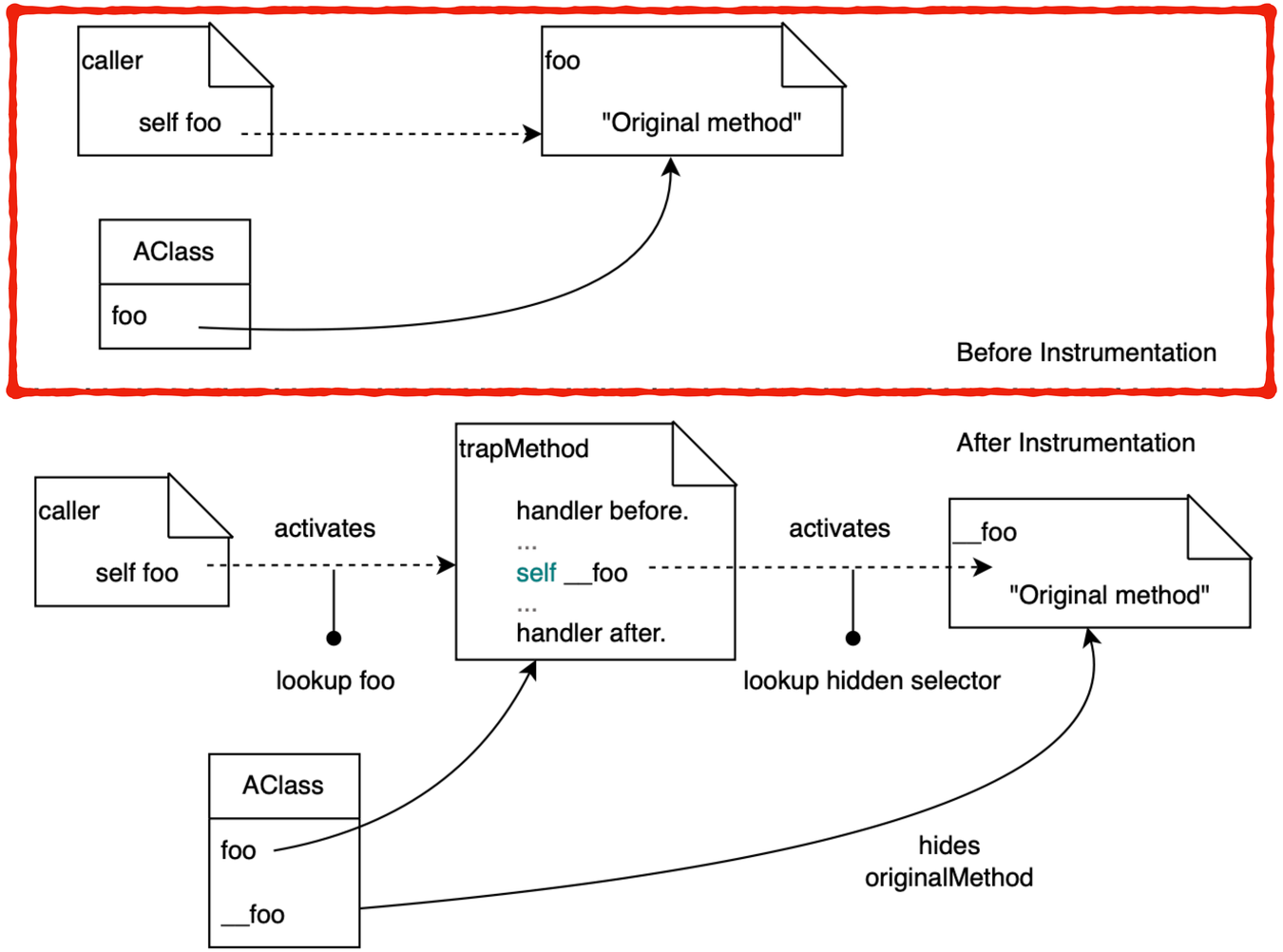
Original code

run:with:in

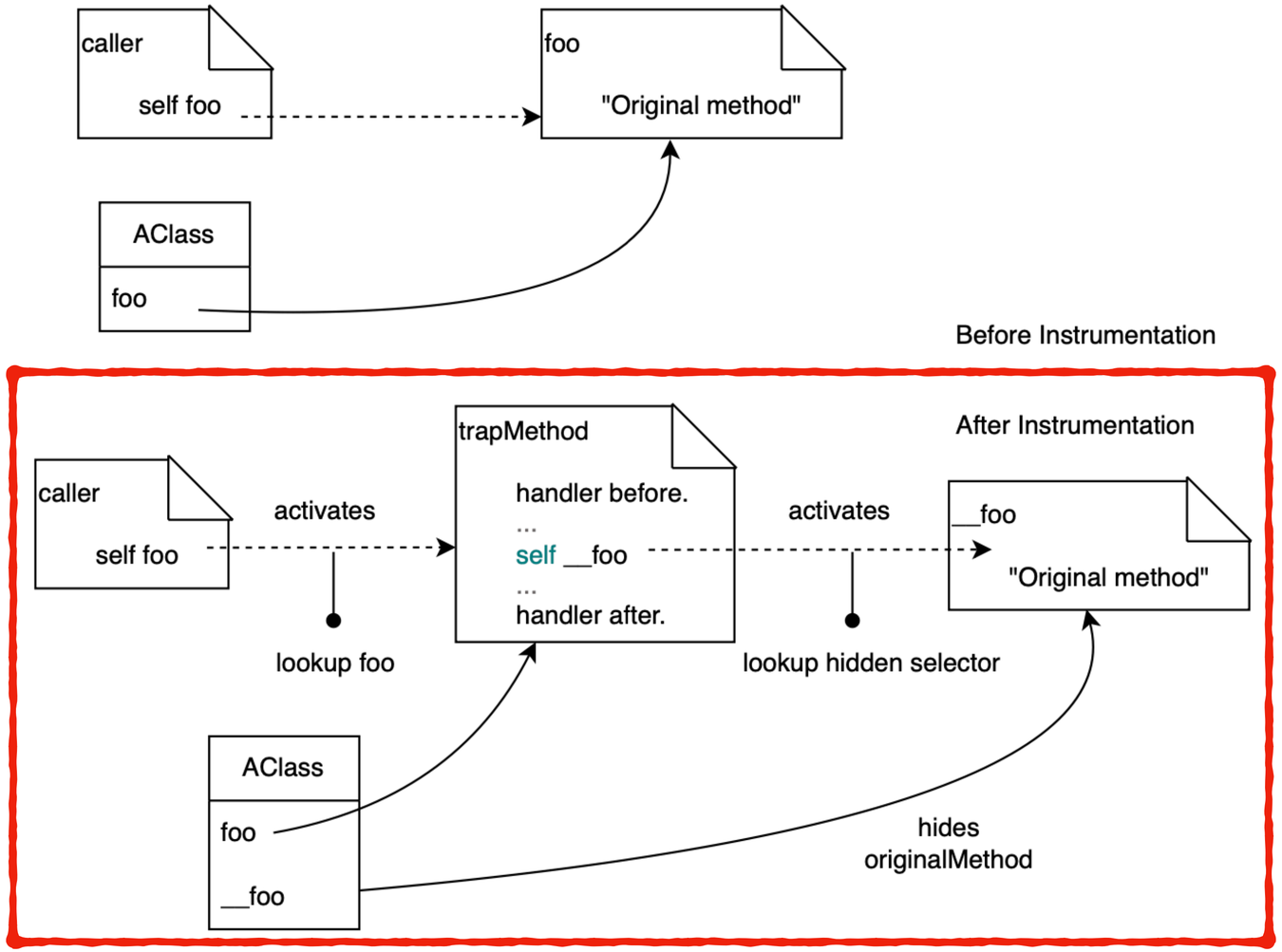
```
ProxyObject >> run: selector with: args in: aReceiver
        | v |
        self isMetaForActiveProcess ifFalse: [
                self runInMetaLevel: [ #beforeHandler ] ].

        v := originalMethod valueWithReceiver: aReceiver arguments: args
        ] ensure: [
            self isMetaForActiveProcess ifFalse: [
                    self runInMetaLevel: [ #afterHandler ] ] ]
        ^ v
```

Meta checking
+ After action

MethodProxies



MethodProxies



Trap Method

```
AClass>> foo: args
	"This is not a primitive, just a marker"
	<primitive: 198>
	"The unwind handler should be the first temp.
	The complete flag should be the second temp."
	| deactivator complete result |
	deactivator := #deactivator.
	#beforeHandler.
	result := self ___foo: args.
	#afterHandler.
	"Mark the execution as complete to avoid double
	execution of the unwind handler"
	complete := true.
	^ result
```

Trap Method

```
AClass>> foo: args
	"This is not a primitive, just a marker"
	<primitive: 198>
	"The unwind handler should be the first temp.
	The complete flag should be the second temp."
	| deactivator complete result |
	deactivator := #deactivator.
	#beforeHandler.
	result := self ___foo: args.
	#afterHandler.
	"Mark the execution as complete to avoid double
	execution of the unwind handler"
	complete := true.
	^ result
```

Trap Method

```
AClass>> foo: args
    "This is not a primitive, just a marker"
    <primitive: 198>
    "The unwind handler should be the first temp.
    The complete flag should be the second temp."
    | deactivator complete result |
    deactivator := #deactivator.
    #beforeHandler.
    result := self ___foo: args.
    #afterHandler.
    "Mark the execution as complete to avoid double
    execution of the unwind handler"
    complete := true.
    ^ result
```

Trap Method

```
AClass>> foo: args
    "This is not a primitive, just a marker"
    <primitive: 198>
    "The unwind handler should be the first temp.
    The complete flag should be the second temp."
    | deactivator complete result |
    deactivator := #deactivator.
    #beforeHandler.
    result := self ___foo: args.
    #afterHandler.
    "Mark the execution as complete to avoid double
    execution of the unwind handler"
    complete := true.
    ^ result
```

Experimental Setup

- RQ1 - Instrumentation and uninstrumentation overhead
- RQ2 - Execution overhead

Projects under analysis

- We define as a **benchmark** the execution of a project's test suites.

Project's name	Description	# methods	# tests
Compression	It provides compression and decompressing utilities.	387	29
File System Manager	Pharo's file system manager.	1426	450
Microdown	A markup language based on Markdown.	1041	472
AST	Pharo's abstract syntax tree (AST) representation.	1591	641

Analysis Tools Under Study

- Method call graph
- Method coverage
- No-action instrumentation

Techniques Under Analysis

- MethodProxies
- #run:with:in
- Source code modification

Benchmark Metrics

- **Overhead time**

$$Overhead = I \ / \ NI$$

- **Instrumentation overhead**

$$Instrumentation\ Overhead = insTime \ / \ lowInsTime$$

- **Uninstrumentation overhead**

$$Uninstrumentation\ Overhead = uninsTime \ / \ lowUninsTime$$

RQ1 Instrumentation Overhead

Instrumentation overhead in milliseconds

	MethodProxies	run:with:in:	Source code modification
		No-action tool	
FileSystem	2.13 × ±0.03	1.43 × ±0.01	20.78 × ±0.07
Microdown	1.64 × ±0.02	1.15 × ±0.01	12.46 × ±0.06
Compression	1.17 × ±0.03	**1.0 × ±0.0**	6.36 × ±0.0
AST	2.37 × ±0.03	1.53 × ±0.04	148.92 × ±2.02
		Call graph tool	
FileSystem	2.14 × ±0.0	1.43 × ±0.01	25.5 × ±0.12
Microdown	1.63 × ±0.02	1.16 × ±0.03	13.3 × ±0.08
Compression	1.17 × ±0.04	**1.0 × ±0.0**	6.65 × ±0.03
AST	2.38 × ±0.03	1.57 × ±0.0	282.37 × ±8.68
		Method coverage tool	
FileSystem	2.14 × ±0.0	1.43 × ±0.0	22.81 × ±0.1
Microdown	1.65 × ±0.01	1.16 × ±0.03	13.04 × ±0.08
Compression	1.16 × ±0.03	**1.0 × ±0.0**	6.57 × ±0.02
AST	2.36 × ±0.01	1.56 × ±0.02	199.28 × ±3.67

Best time for all cases

RQ1 Instrumentation Overhead

Instrumentation overhead in milliseconds

	MethodProxies	run:with:in:	Source code modification
	No-action tool		
FileSystem	2.13 × ±0.03	1.43 × ±0.01	20.78 × ±0.07
Microdown	1.64 × ±0.02	1.15 × ±0.01	12.46 × ±0.06
Compression	1.17 × ±0.03	**1.0 × ±0.0**	6.36 × ±0.0
AST	2.37 × ±0.03	1.53 × ±0.04	148.92 × ±2.02
	Call graph tool		
FileSystem	2.14 × ±0.0	1.43 × ±0.01	25.5 × ±0.12
Microdown	1.63 × ±0.02	1.16 × ±0.03	13.3 × ±0.08
Compression	1.17 × ±0.04	**1.0 × ±0.0**	6.65 × ±0.03
AST	2.38 × ±0.03	1.57 × ±0.0	282.37 × ±8.68
	Method coverage tool		
FileSystem	2.14 × ±0.0	1.43 × ±0.0	22.81 × ±0.1
Microdown	1.65 × ±0.01	1.16 × ±0.03	13.04 × ±0.08
Compression	1.16 × ±0.03	**1.0 × ±0.0**	6.57 × ±0.02
AST	2.36 × ±0.01	1.56 × ±0.02	199.28 × ±3.67

Worst time for all cases

RQ1 Instrumentation Overhead

Instrumentation overhead in milliseconds

	MethodProxies	run:with:in:	Source code modification
		No-action tool	
FileSystem	2.13 × ±0.03	1.43 × ±0.01	20.78 × ±0.07
Microdown	1.64 × ±0.02	1.15 × ±0.01	12.46 × ±0.06
Compression	1.17 × ±0.03	**1.0 × ±0.0**	6.36 × ±0.0
AST	2.37 × ±0.03	1.53 × ±0.04	148.92 × ±2.02
		Call graph tool	
FileSystem	2.14 × ±0.0	1.43 × ±0.01	25.5 × ±0.12
Microdown	1.63 × ±0.02	1.16 × ±0.03	13.3 × ±0.08
Compression	1.17 × ±0.04	**1.0 × ±0.0**	6.65 × ±0.03
AST	2.38 × ±0.03	1.57 × ±0.0	282.37 × ±8.68
		Method coverage tool	
FileSystem	2.14 × ±0.0	1.43 × ±0.0	22.81 × ±0.1
Microdown	1.65 × ±0.01	1.16 × ±0.03	13.04 × ±0.08
Compression	1.16 × ±0.03	**1.0 × ±0.0**	6.57 × ±0.02
AST	2.36 × ±0.01	1.56 × ±0.02	199.28 × ±3.67

Close to #run:with:in

RQ1 Uninstrumentation Overhead

Uninstrumentation overhead in milliseconds

	MethodProxies	run:with:in:	Source code modification
	No-action tool		
FileSystem	1.78 × ±0.01	1.14 × ±0.0	1.65 × ±0.02
Microdown	1.17 × ±0.04	1.08 × ±0.02	1.3 × ±0.03
Compression	**1.0 × ±0.0**	1.07 × ±0.0	1.07 × ±0.0
AST	1.92 × ±0.02	1.21 × ±0.0	1.93 × ±0.01
	Call graph tool		
FileSystem	1.86 × ±0.02	1.21 × ±0.0	1.58 × ±0.02
Microdown	1.29 × ±0.0	1.21 × ±0.0	1.35 × ±0.02
Compression	**1.07 × ±0.0**	1.15 × ±0.01	1.14 × ±0.02
AST	2.0 × ±0.0	1.29 × ±0.02	1.92 × ±0.02
	Method coverage tool		
FileSystem	1.85 × ±0.01	1.21 × ±0.0	1.5 × ±0.0
Microdown	1.24 × ±0.03	1.15 × ±0.02	1.27 × ±0.03
Compression	**1.07 × ±0.0**	1.14 × ±0.0	1.12 × ±0.03
AST	1.94 × ±0.02	1.29 × ±0.0	1.79 × ±0.01

RQ1 Conclusion

- **Instrumentation**
  - MethodProxies incurs an instrumentation overhead ranging from 1.16 to 2.38 × compared to the fastest time of run:with:in:
  - It is significantly faster than the source code modification technique.
- **Uninstrumentation**
  - There are no big differences among all the techniques

RQ2 Execution Overhead

Overhead for executing the instrumented code

	MethodProxies	run:with:in:	Source code modification
	No-action tool		
FileSystem	1.03 × ±0.0	1.17 × ±0.0	1.08 × ±0.0
Microdown	**0.91 × ±0.1**	17.98 × ±1.05	6.14 × ±0.16
Compression	1.05 × ±0.0	9.33 × ±0.22	3.31 × ±0.01
AST	5.15 × ±0.05	47.92 × ±2.75	23.33 × ±0.06
	Call graph tool		
FileSystem	**1.07 × ±0.0**	1.22 × ±0.0	1.11 × ±0.0
Microdown	4.35 × ±0.2	20.87 × ±1.13	8.7 × ±0.16
Compression	2.49 × ±0.01	10.56 × ±0.13	4.35 × ±0.01
AST	25.48 × ±0.23	49.87 × ±0.77	37.62 × ±0.15
	Method coverage tool		
FileSystem	**1.05 × ±0.0**	1.19 × ±0.0	1.09 × ±0.0
Microdown	2.22 × ±0.11	19.18 × ±0.8	6.82 × ±0.15
Compression	1.58 × ±0.01	9.73 × ±0.23	3.59 × ±0.01
AST	11.61 × ±0.13	44.08 × ±0.62	28.89 × ±0.09

Fastest in all cases

RQ2 Execution Overhead

Overhead for executing the instrumented code

	MethodProxies	run:with:in:	Source code modification
	No-action tool		
FileSystem	1.03 × ±0.0	1.17 × ±0.0	1.08 × ±0.0
Microdown	**0.91 × ±0.1**	17.98 × ±1.05	6.14 × ±0.16
Compression	1.05 × ±0.0	9.33 × ±0.22	3.31 × ±0.01
AST	5.15 × ±0.05	47.92 × ±2.75	23.33 × ±0.06
	Call graph tool		
FileSystem	**1.07 × ±0.0**	1.22 × ±0.0	1.11 × ±0.0
Microdown	4.35 × ±0.2	20.87 × ±1.13	8.7 × ±0.16
Compression	2.49 × ±0.01	10.56 × ±0.13	4.35 × ±0.01
AST	25.48 × ±0.23	49.87 × ±0.77	37.62 × ±0.15
	Method coverage tool		
FileSystem	**1.05 × ±0.0**	1.19 × ±0.0	1.09 × ±0.0
Microdown	2.22 × ±0.11	19.18 × ±0.8	6.82 × ±0.15
Compression	1.58 × ±0.01	9.73 × ±0.23	3.59 × ±0.01
AST	11.61 × ±0.13	44.08 × ±0.62	28.89 × ±0.09

RQ2 Conclusion

- Among all benchmarks and analysis tools, **MethodProxies has the lowest execution overhead**

More in the paper!

MethodProxies: A Safe and Fast Message-Passing Control Library

Sebastian Jordan Montaño[1], Juan Pablo Sandoval Alcocer[2], Guillermo Polito[1], Stéphane Ducasse[1] and Pablo Tesone[1]

[1]*Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, Park Plaza, Parc scientifique de la Haute-Borne, 40 Av. Halley Bât A, 59650 Villeneuve-d'Ascq, France*

[2]*Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile*

Abstract

The injection of monitoring code allows for real-time observation of the program, which has proven instrumental in developing tools that assist developers with various programming tasks. In dynamic languages such as Pharo, renowned for their rich meta-programming capabilities and dynamic method dispatch, such monitoring capabilities are particularly valuable. Message-passing control techniques are commonly used to monitor program execution at the method level, involving the execution of specific code before and after each method invocation. Implementing message-passing control techniques, however, poses many challenges, notably in terms of instrumentation overhead. Additionally, it is crucial for the message-passing mechanism to be safe: *i.e.,* to accommodate recursive and reflective scenarios to ensure that it does not alter the execution of the monitored program, which could potentially lead to infinite loops or other unintended consequences.

Over the years, numerous techniques have been proposed to optimize message-passing control. This paper introduces MethodProxies, a message-passing instrumentation library that offers minimal overhead and is safe. We conduct a comparison between MethodProxies and two commonly used techniques implemented in the Pharo programming language: method substitution using the run:with:in: hook and source code modification. Our results demonstrate that MethodProxies offers significantly lower overhead compared to the other two techniques and is safe against infinite recursion.

Keywords

instrumentation, message-passing control, error handling, method compilation

- More implementation details
- Detailed research questions
- Discussion and threats to validity

MethodProxies: a Safe and Fast Library

- MethodProxies is safe and fast
- MethodProxies has the **lowest execution overhead**
- It allows to instrument **any method** on the system
- We use it on several tools and applications to make the instrumentation

 github.com/pharo-contributions/MethodProxies

Sebastian JORDAN MONTAÑO
sebastian.jordan@inria.fr