Object-Centric Instrumentation with Pharo

Steven Costiou

Layout and typography based on the sbabook LaTeX class by Damien Pollet.

Contents

Illustrations

Introduction

This booklet is about object-centric instrumentation in Pharo. An instrumentation is object-centric if it applies to one specific object (or a set of objects), without consideration of its class. It means the instrumentation can be applied on one object, leaving untouched all other instances of its class, or to an heterogeneous set of instances of different classes. This booklet gives an overview of available object-centric instrumentation techniques in Pharo, either present in the standard distribution or available on download. We will not go into deep technical usage description, nor into implementation details. Each chapter illustrates one solution with examples, and gives the necessary references if one wants to go deeper in the study of the solution. We study each technique following a three-fold evaluation. First, the studied technique is applied on a simple example of object-centric instrumentation. Second, the technique is evaluated against a set of desirable properties. Finally, performance overhead are evaluated. Only the raw solution is evaluated, without considering the possibility of enhancing the technique by building something on top.

1.1 Illustration example

Each studied solution is experimented on a trivial example of object-centric behavior instrumentation. This example is illustrated in script 1-1. Two instances of `OrderedCollection` are created, and to each of these instances is sent the `add:` message with a string as a parameter. The instrumentation must happen in-between. We want to instrument the `col1` object, so that when the `add:` message is received, the size of the collection and the added object (passed as parameter) are printed in the `Transcript`. The `col2` object must remain unaffected.

Listing 1-1   Trivial example for object-centric instrumentation

```
|col1 col2|
col1 := OrderedCollection new.
col2 := OrderedCollection new.

"...instrumentation must happen here..."

col1 add: 'Hello World'.
col2 add: 'Hello World'.
```

1.2   Evaluation criteria

Each solution is evaluated against the following desirable properties.

Property	Definition
Manipulated entity	The unit of instrumentation (*e.g.* a class, a Trait, an object...)
Reusability	The entity can be reused to instrument different objects
Flexibility	Instrumentation does not put constraint on the source code or in the coding style
Granularity	The level of at which behavior can be instrumented (*e.g.* method, AST...)
Integration	Instrumentation does not break system features

1.3   Performance overhead evaluation

To provide a approximation of the performance overhead due to instrumentation, we compare the execution time of a block of code without instrumentation with the execution time of an instrumented block of code. Script 1-2 shows how the average execution time is computed. An ordered collection is instantiated (`collection`) and the `add:` message is sent 1000 times to that object. Before actually sending that message to the collection, the parameter is logged in the Transcript. Each time, the execution time of the parameter logging and the `add:` method is stored into the `execTime` collection. The average execution time is computed from the stored results. One reference evaluation is performed without instrumentation. This reference average time is used for comparison with instrumented versions evaluations. For each studied technique, the same code is executed, but with an object-centric instrumentation put on the `collection` object. However, that time the logging instruction of the `add:` mesage parameter is removed from the script, and added as an object-centric instrumentation on the `add:` method of the collection object. We suppose that the execution time of this instruction is constant, so that we can isolate the cost of the instrumentation mechanism and compare it against the cost of having the instrumentation code hard-coded.

Listing 1-2   Performance overhead evaluation script

```
|execTimes collection|
execTimes := OrderedCollection new.
collection := OrderedCollection new.
1 to: 1000 do:[:i|
  execTimes add: [
    i logCr. "That line is removed for instrumented evaluations"
    collection add: i] timeToRun].
execTimes average
```

1.4   Structure of the book

The second chapter will provide an overview of the evaluation results of object-centric instrumentation techniques available in Pharo. A reader may directly read this chapter if he is already familiar with the Pharo techniques presented in the book. Chapters 3 to 7 describe five solutions for object-centric instrumentation, and provide an evaluation of these solutions. Chapter 8 drafts the premises of an object-centric debugger and concludes the book.

Summary of the overall evaluations

If you already know Pharo and (some of) the presented technique, this chapter is a global summary with spoilers.
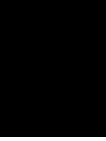
Anonymous subclasses

3.1 **What are Talents**

3.2 **Example**

Installing Talents

aa

Example

bb

Listing 3-1 Installation from Github

```
Metacello new
  baseline: 'Talents';
  repository: 'github://tesonep/pharo-talents/src';
  load.
```

Listing 3-2 Installation from Github

```
talent := Trait named: 'MyTalent'.
talent compile: 'add: anObject
anObject logCr.
super add: anObject'.
col := OrderedCollection new.
col addTalent: talent.
col add: 'This is an added object.'
```

3.3 **Evaluation**

cc

▎ **Note**   this is a note annotation.

▎ **To do**   this is a todo annotation

Country	Capital
France	Paris
Belgium	Brussels
Country	**Capital**
France	Paris
Belgium	Brussels

4

Talents

Talents [RGN$^+$14] is this.

4.1 What are Talents

4.2 Example

Installing Talents

aa

Example

bb

4.3 Evaluation

cc

Listing 4-1 Installation from Github

```
Metacello new
  baseline: 'Talents';
  repository: 'github://tesonep/pharo-talents/src';
  load.
```

Listing 4-2 Installation from Github

```
talent := Trait named: 'MyTalent'.
talent compile: 'add: anObject
anObject logCr.
super add: anObject'.
col := OrderedCollection new.
col addTalent: talent.
col add: 'This is an added object.'
```

❚ **Note** this is a note annotation.

❚ **To do** this is a todo annotation

Country	Capital
France	Paris
Belgium	Brussels
Country	**Capital**
France	Paris
Belgium	Brussels

Proxies

5.1 **What are Talents**

5.2 **Example**

Installing Talents

aa

Example

bb

Listing 5-1   Installation from Github

```
Metacello new
  baseline: 'Talents';
  repository: 'github://tesonep/pharo-talents/src';
  load.
```

Listing 5-2   Installation from Github

```
talent := Trait named: 'MyTalent'.
talent compile: 'add: anObject
anObject logCr.
super add: anObject'.
col := OrderedCollection new.
col addTalent: talent.
col add: 'This is an added object.'
```

5.3 **Evaluation**

cc

❚ **Note**   this is a note annotation.

❚ **To do**   this is a todo annotation

Country	Capital
France	Paris
Belgium	Brussels
Country	**Capital**
France	Paris
Belgium	Brussels

Reflectivity

Talents [RGN$^+$14]is this.

6.1 **What are Talents**

6.2 **Example**

Installing Talents

aa

Example

bb

6.3 **Evaluation**

cc

Listing 6-1   Installation from Github

```
Metacello new
  baseline: 'Talents';
  repository: 'github://tesonep/pharo-talents/src';
  load.
```

Listing 6-2   Installation from Github

```
talent := Trait named: 'MyTalent'.
talent compile: 'add: anObject
anObject logCr.
super add: anObject'.
col := OrderedCollection new.
col addTalent: talent.
col add: 'This is an added object.'
```

❚ **Note**   this is a note annotation.

❚ **To do**   this is a todo annotation

Country	Capital
France	Paris
Belgium	Brussels
Country	**Capital**
France	Paris
Belgium	Brussels

Low-level techniques

7.1 Example

Installing Talents

aa

Example

bb

Listing 7-1 Installation from Github

```
Metacello new
  baseline: 'Talents';
  repository: 'github://tesonep/pharo-talents/src';
  load.
```
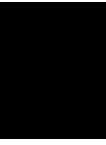
Listing 7-2 Installation from Github

```
talent := Trait named: 'MyTalent'.
talent compile: 'add: anObject
anObject logCr.
super add: anObject'.
col := OrderedCollection new.
col addTalent: talent.
col add: 'This is an added object.'
```

7.2 Evaluation

cc

❚ **Note**　this is a note annotation.

❚ **To do**　this is a todo annotation

Country	Capital
France	Paris
Belgium	Brussels
Country	**Capital**
France	Paris
Belgium	Brussels

Conclusion

8.1 **Example**

Installing Talents

aa

Example

bb

Listing 8-1   Installation from Github

```
Metacello new
  baseline: 'Talents';
  repository: 'github://tesonep/pharo-talents/src';
  load.
```

Listing 8-2   Installation from Github

```
talent := Trait named: 'MyTalent'.
talent compile: 'add: anObject
anObject logCr.
super add: anObject'.
col := OrderedCollection new.
col addTalent: talent.
col add: 'This is an added object.'
```

8.2 **Evaluation**

cc

▐ **Note**   this is a note annotation.

▐ **To do**   this is a todo annotation

Country	Capital
France	Paris
Belgium	Brussels
Country	**Capital**
France	Paris
Belgium	Brussels

Bibliography

[RGN$^+$14] Jorge Ressia, Tudor Gîrba, Oscar Nierstrasz, Fabrizio Perin, and
Lukas Renggli. Talents: an environment for dynamically compos-
ing units of reuse. *Software: Practice and Experience*, 44(4):413–432,
2014.