Object-Centric Instrumentation with Pharo

Steven Costiou

Square Bracket tutorials
February 21, 2019

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

Contents

Illustrations

CHAPTER 1

Ghost

Ghost is a general and uniform proxy implementation[PBF+15]. A proxy replaces an object to control access to that object [ABW98]. Object-centric instrumentation by means of proxies is done by swapping an object (and all its references) with a proxy object (and references to that proxy object). A proxy object is instance of a proxy class, in which access control is defined. Access control is generally implemented through a single interface, which is called each time a message is intercepted by the proxy. Control behavior then decides what to do with the received message.

1.1 Example

In this example, we use the original implementation of Ghost [PBF+15].

```
GHProxyHandler subclass: #MyProxyHandler
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'ObjectCentricEvaluationExamples'

MyProxyHandler>>manageMessage: interception
  | message proxy target result |
  message := interception message.
  proxy := interception proxy.
  target := proxy proxyTarget.
  message selector == #name:
    ifTrue: [ target tag: message arguments first ].
  result := message sendTo: target.
  ^ result == target
    ifTrue: [ proxy ]
    ifFalse: [ result ]
```

```
17
18 MyProxyHandler>>handleUninstall: anInterception
19   ^ anInterception proxy proxyTarget become: anInterception proxy.
```

Then we instrument the objects

```
1 |person|
2   person := Person new.
3   GHTargetBasedProxy createProxyAndReplace: person handler:
      MyProxyHandler new.
4   person uninstall
```

1.2 Evaluation

Manipulated entity: Classes. Proxies are defined and/or configured in classes which inherits from Ghost internal classes. Typically, developers subclass the base message handler from Ghost to create a proxy model that implements the wanted instrumentation. An API is provided to apply a proxy to objects.

Reusability: Complete. The same proxy model can be reused to instrument any kind of object with the same instrumentation. Although Ghost proxies are not meant to be composed, it should be possible to *proxify* a proxy, if a proper model is implemented to instrument a proxy object.

Flexibility: Partial. Because the user has to define a proxy model which specifies which messages are handled and/or which are not. However that allows developers to fully and transparently integrate the instrumented object into the environment.

Granularity: Method. A proxy intercepts messages sends to the object it *proxifies*. It can execute instrumentation behavior before, after or instead the intercepted message. Sub-method instrumentation cannot be achieved by means of proxies.

Integration: Full. because meta-messages can bde efined

`Self` problem: Implementation dependent.

`Super` problem: Implementation dependent. ?

1.3 Other documentation

Implementations and documentation based on the original Ghost paper [PBF+15]:

- http://esug.org/data/ESUG2011/IWST/PRESENTATIONS/23.Mariano_Peck-Ghost-ESUG2011.pdf
- https://rmod.inria.fr/archives/papers/Mart14z-Ghost-Final.pdf

- https://gitlab.inria.fr/RMOD/Ghost
- https://github.com/guillep/avatar

Another implementation of Ghost:

- https://github.com/pharo-ide/Ghost
- http://dionisiydk.blogspot.com/2016/04/halt-next-object-call.html

Bibliography

[ABW98] Sherman R Alpert, Kyle Brown, and Bobby Woolf. *The design patterns Smalltalk companion*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[PBF+15] Mariano Martinez Peck, Noury Bouraqadi, Luc Fabresse, Marcus Denker, and Camille Teruel. Ghost: A uniform and general-purpose proxy implementation. *Journal of Object Technology*, 98:339–359, 2015.