Pharo: a reflective language
A first systematic analysis of reflective APIs

Iona Thomas, Stéphane Ducasse, Pablo Tessone, Guillermo Polito

IWST 2023 - 29/08/2023

What is reflection ?

“Reflection is the ability of a program to manipulate as data something representing the state of the program during its own execution.” [1]

1. R.G. Gabriel and D.G. Bobrow and J.L. White. CLOS in Context - The Shape of the Design Space. In Object OrientedProgramming - The CLOS perspective. MIT Press, 1993.

Reflection by example

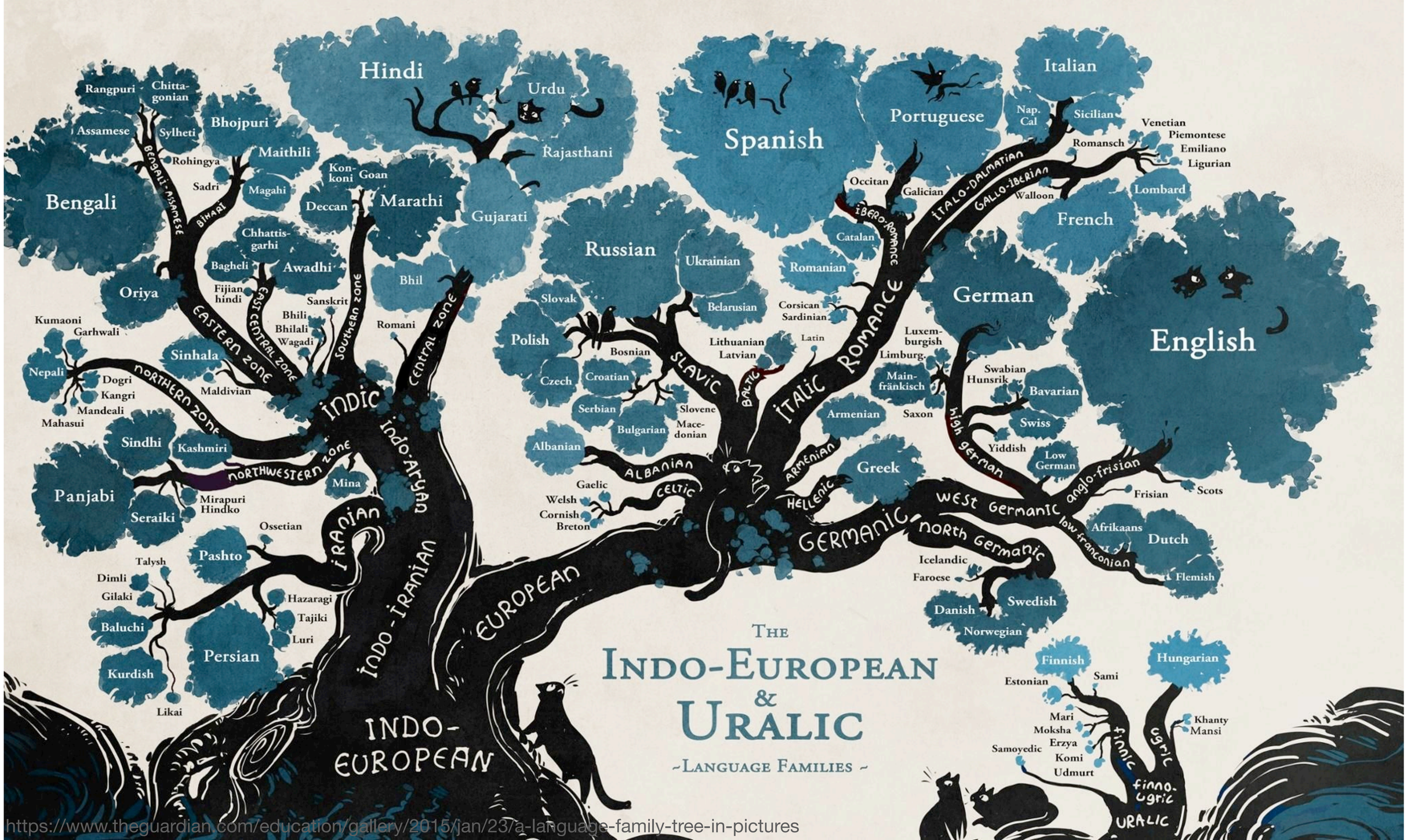
```
instVarNamed: aString put: aValue
  "Store into the value of the instance variable in me of that name. Slow, but very useful.
  We support here all slots (even non indexed) but raise a backward compatible exception"

  ^ self class
    slotNamed: aString
    ifFound: [ :slot | slot write: aValue to: self ]
    ifNone: [ InstanceVariableNotFound signalFor: aString asString ]
```

Reflection categories

Often separated in 3 subcategories :

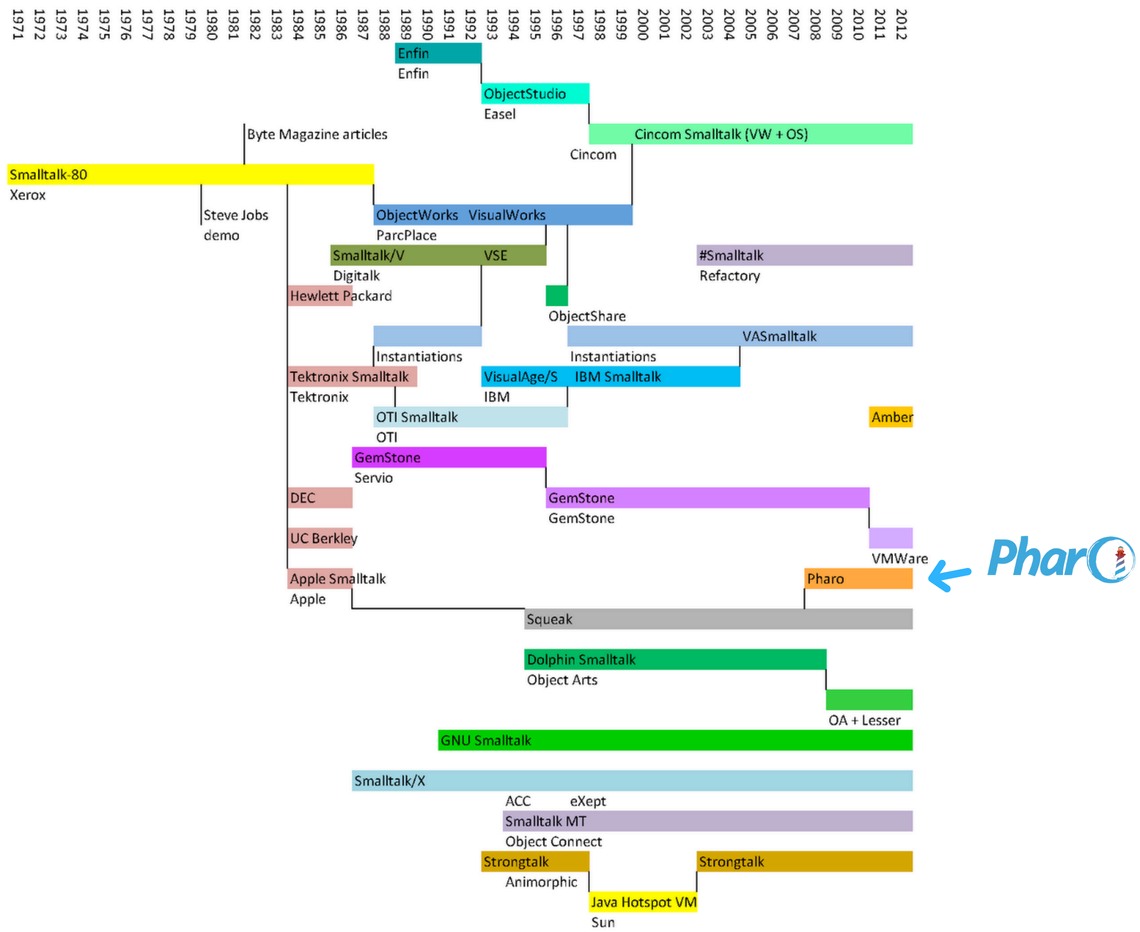
- Introspection
- Self-modification
- Intercession



https://www.theguardian.com/education/gallery/2015/jan/23/a-language-family-tree-in-pictures

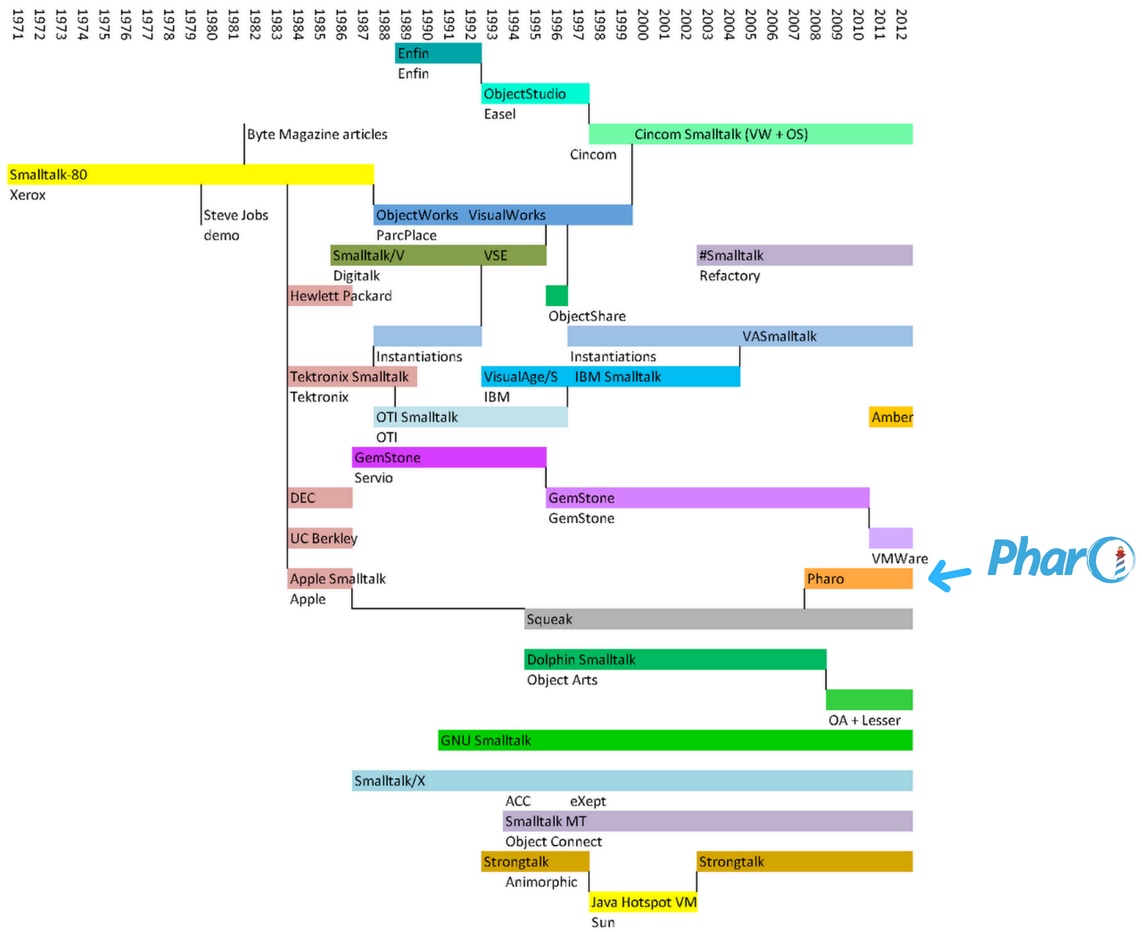
Why an analysis of reflective features ?

Language evolution :



Why an analysis of reflective features ?

Language evolution :

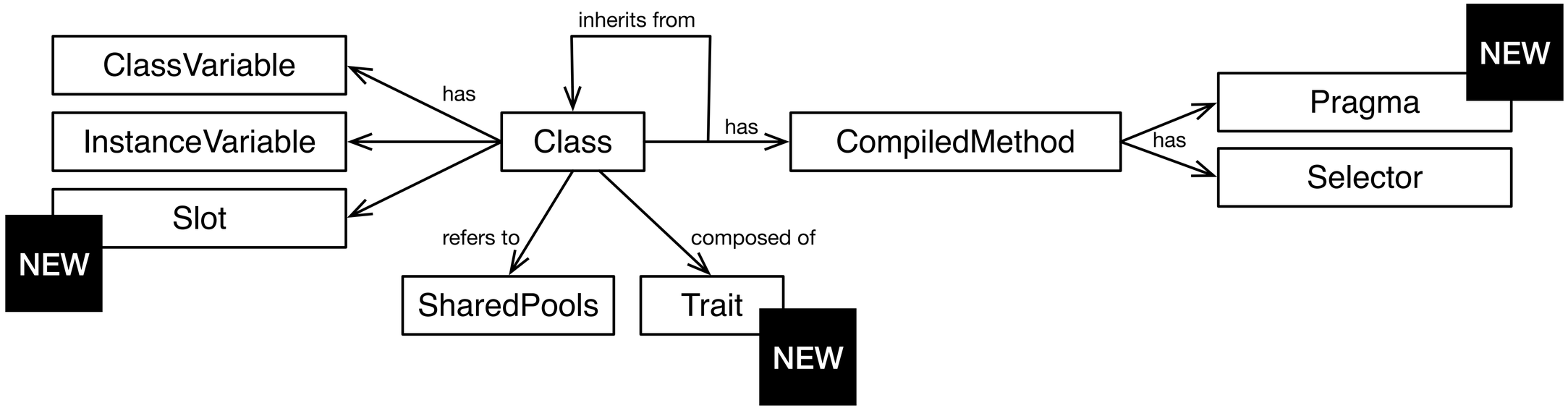


Understand current state of reflection :

- Years of **organic evolution**
- **Outdated documentations** :
  - *Smalltalk-80: The Language and Its Implementation* (83) (Blue book)
  - *Smalltalk: a Reflective Language* (96)
  - ANSI Smalltalk (98)
- **New concepts** : Pragmas, Slots, Packages …

Why an analysis of reflective features ?

Evolution of the structural pharo meta-model



Why an analysis of reflective features ?

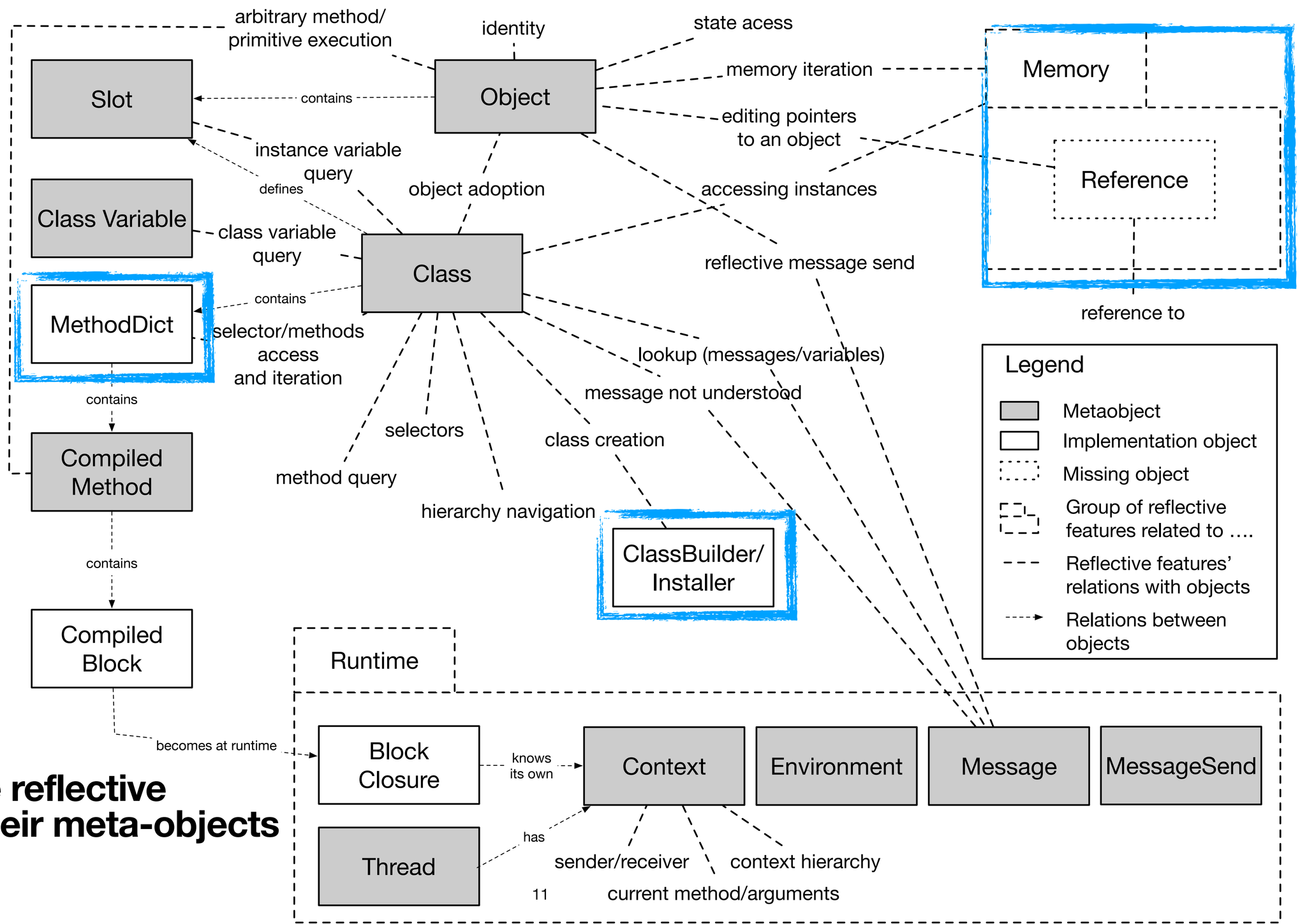
Once we understand the current state, we can :

- (Re)design the MOP
- Modularize some of the reflective API
- Study reflection from security/stability point of view

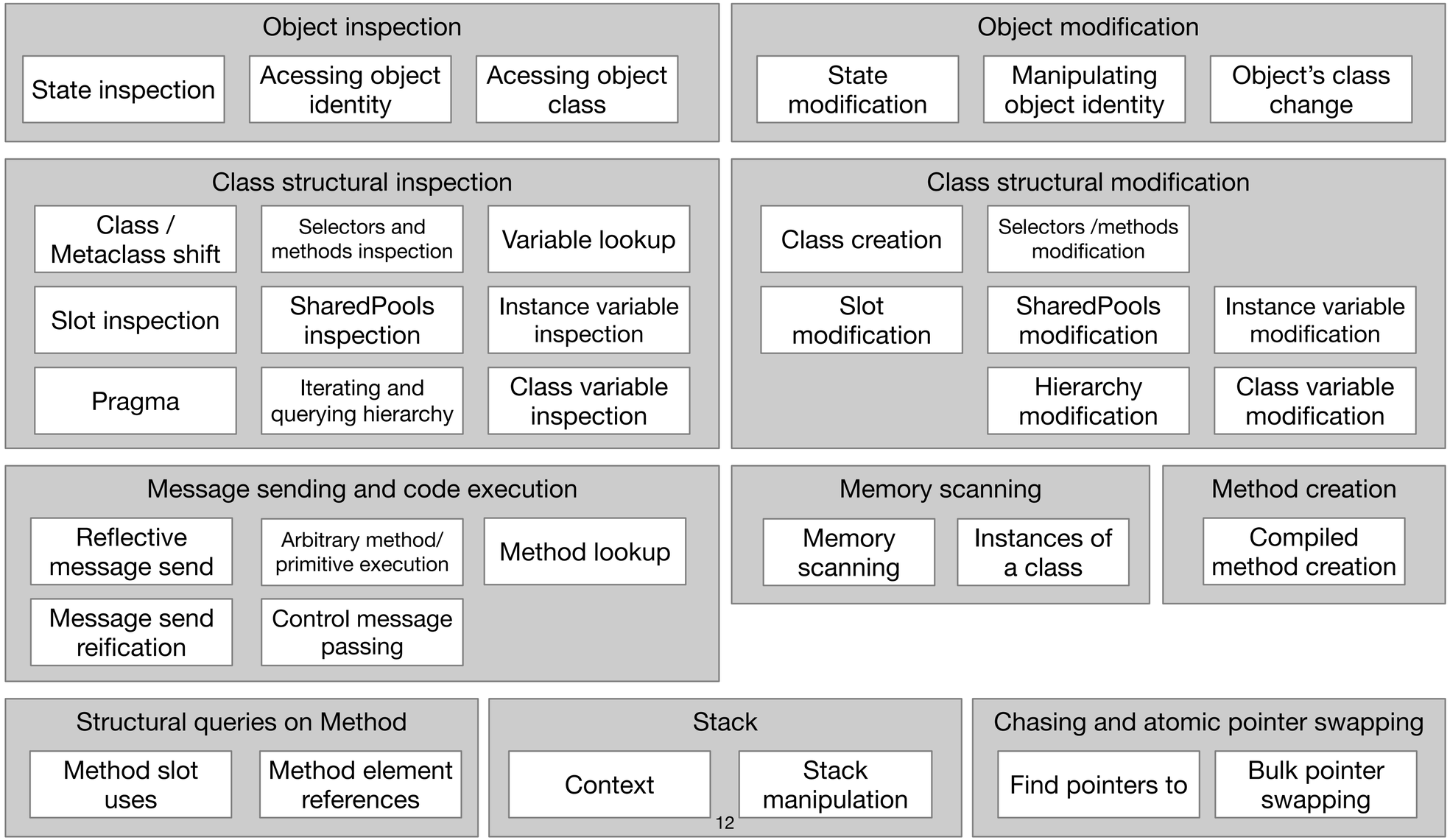
Goals

- Create a catalog of reflective API
- Analyse meta-objects
- Analyse reflective APIs
- Highlight areas of improvements

A map of some reflective features and their meta-objects

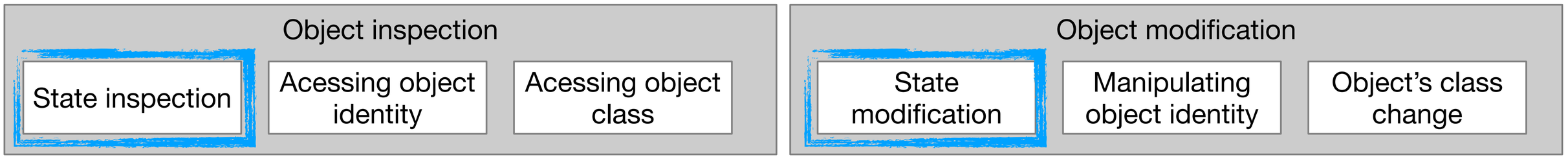


Categories


Object inspection
State inspection
Acessing object identity
Acessing object class
Object modification
State modification
Manipulating object identity
Object's class change
Class structural inspection
Class / Metaclass shift
Selectors and methods inspection
Variable lookup
Slot inspection
SharedPools inspection
Instance variable inspection
Pragma
Iterating and querying hierarchy
Class variable inspection
Class structural modification
Class creation
Selectors /methods modification
Slot modification
SharedPools modification
Instance variable modification
Hierarchy modification
Class variable modification
Message sending and code execution
Reflective message send
Arbitrary method/ primitive execution
Method lookup
Message send reification
Control message passing
Memory scanning
Memory scanning
Instances of a class
Method creation
Compiled method creation
Structural queries on Method
Method slot uses
Method element references
Stack
Context
Stack manipulation
Chasing and atomic pointer swapping
Find pointers to
Bulk pointer swapping

Accessing instance variables

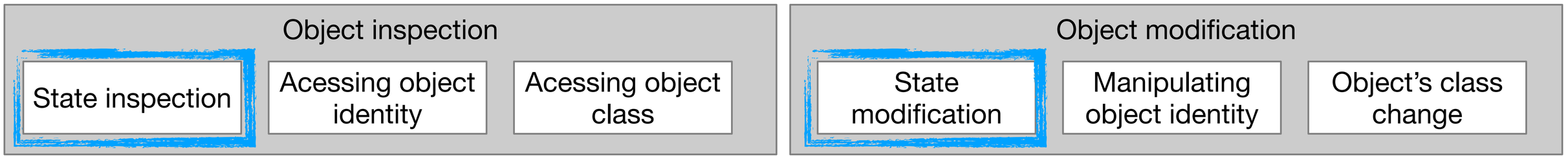
API



- Object>>InstVarAt:
- Object>>InstVarNamed:
- Object>>InstVarAt:Put:
- Object>>InstVarNamed:Put:

Accessing instance variables

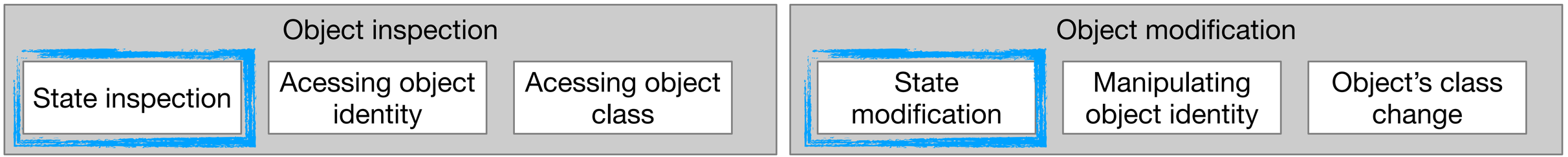
API



- Object>>InstVarAt:
- Object>>InstVarNamed:
- ProtoObject>>InstVarsInclude:
- Context>>objectSize:
- Object>>InstVarAt:Put:
- Object>>InstVarNamed:Put:

Accessing instance variables

Uses and potential improvements

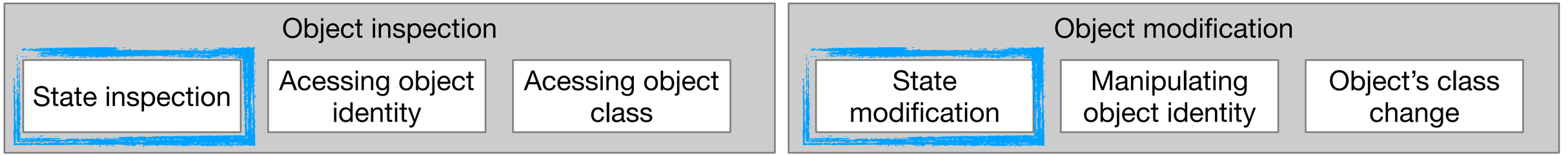


Current uses :

- Copying objects
- Serialising objects
- Printing objects
- Inspecting objects
- Checking references
- Testing, including for some fields without getter methods

Accessing instance variables

Uses and potential improvements



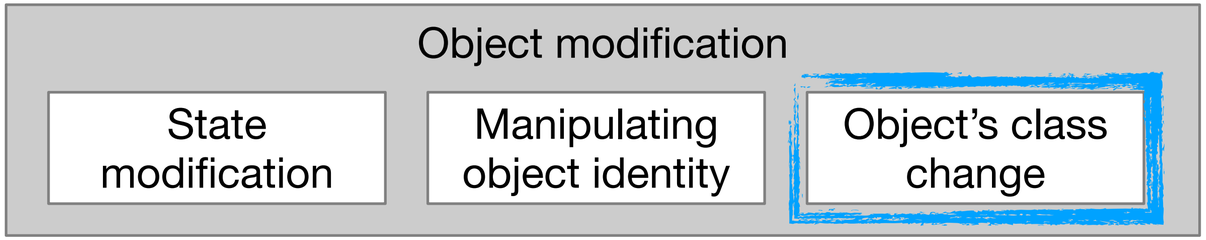
Current uses :

- Copying objects
- Serialising objects
- Printing objects
- Inspecting objects
- Checking references
- Testing, including for some fields without getter methods

Areas of improvement :

- No solution for intercession on state read/write

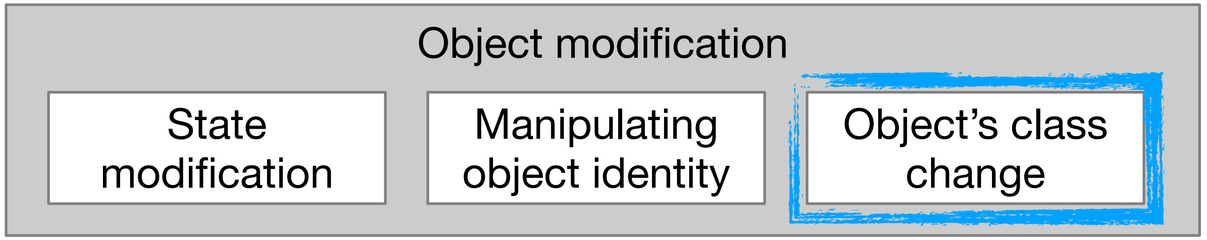
Changing the class of an object



API

- Behavior»adoptInstance:
- Metaclass»adoptInstance:from:
- Object»primitiveChangeClassTo:

Changing the class of an object



API

- Behavior»adoptInstance:
- Metaclass»adoptInstance:from:
- Object»primitiveChangeClassTo:

Uses :

- Proxy implementation
- Updating instances to the new version of a class

Areas of improvement :

- Constraints on same class format
- Some state can be lost (class builder)

What is next ?

- Assess potential safety issues
- Better understand how the reflective feature are used
- Modularisation of some reflective operations

More in the paper !

Pharo: a reflective language – A first systematic analysis of reflective APIs

Iona Thomas[1], Stéphane Ducasse[1], Pablo Tesone[1] and Guillermo Polito[1]

[1]*Univ Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRIStAL.*

Abstract

Reflective operations are powerful APIs that let developers build advanced tools or architecture. Reflective operations are used for implementing tools and development environments (*e.g.,* compiler, debugger, inspector) or language features (*e.g.,* distributed systems, exception, proxy, aspect-oriented programming). In addition, languages are evolving, introducing better concepts, and revising practices and APIs. As such, since 2008 Pharo evolved and was built on Squeak which changed from the original Smalltalk reflective APIs. Pharo has one of the largest reflective feature sets ranging from structural reflection to on-demand stack reification. In addition, new metaobjects got integrated such as first-class instance variables. Finally, reflective facilities are mixed with the base-level API of objects and classes and reflective features are heavily used by the system tools.

Getting an understanding of such API is, however, tedious when the API is large and evolved over a decade. There is a need for a deep analysis of current reflective APIs to understand their underlying use, potential dependencies, and whether some reflective features can be scoped and optional.

In this article, we analyze the reflective operations used in Pharo 11. We classify the current reflective operations in different families. Also, we identify a set of issues raised by the use of reflective operations. Such an analysis of reflective operations in Pharo is important to support the revision of the reflective layer and its potential redesign.

Keywords

Reflection, Meta-object protocols

- Memory Scanning
- Stack Manipulation

… and more !

Key Takeaways

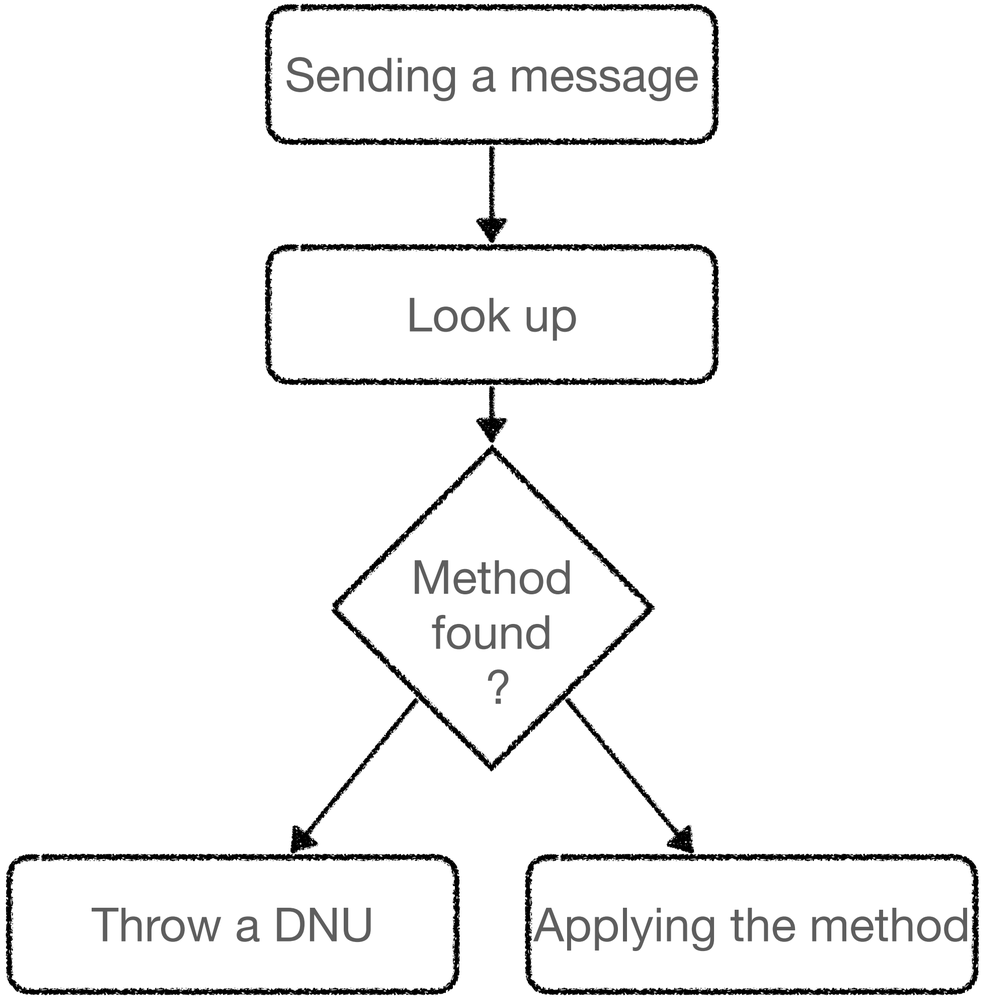
Contributions :

- A Catalog of reflective API
- Analysis on meta-objects
- Analysis on reflective APIs
- Areas of improvements for the reflective API

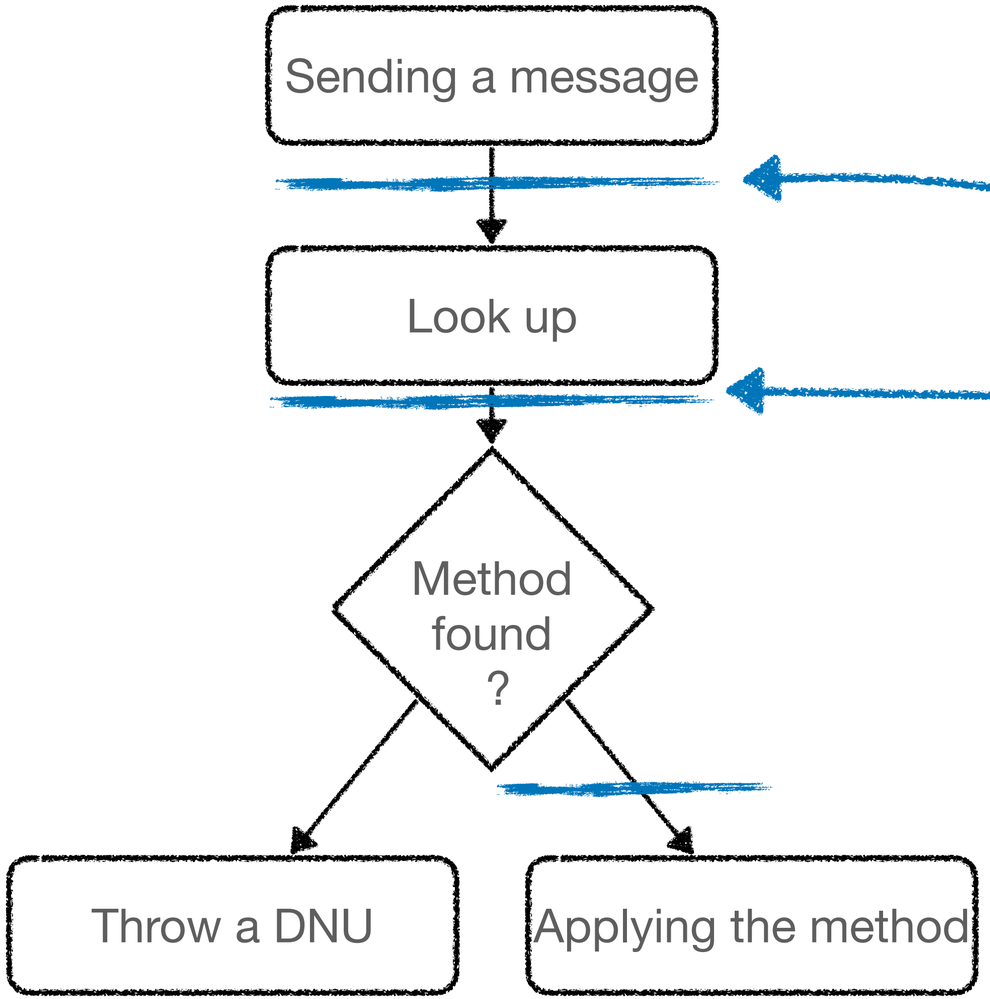
Why read the article ?

- Discovering the reflective API of Pharo
- Digging into Pharo MOP design
- Implementing some of the suggested improvements !


The Limites of Intercession



The Limits of Intercession



No high-level API to do intercession here