A New Architecture

Reconciling Refactorings and Transformations

Balša Šarenac, Stéphane Ducasse and Nicolas Anquetil

Definitions

- Transformation
- Refactoring
- Precondition
- Atomic refactoring
- Composite refactoring

Imagine a world with only refactorings

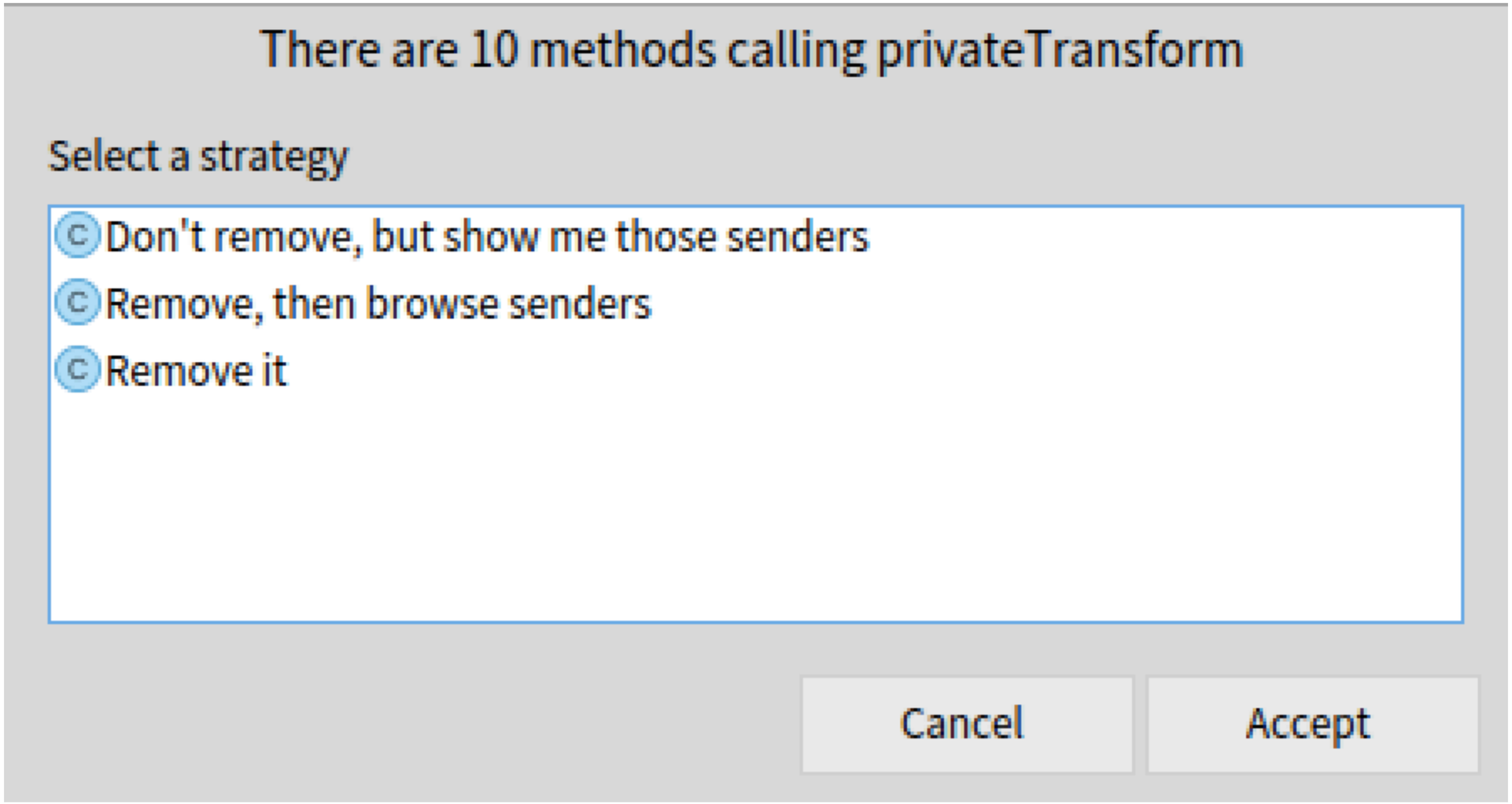
- Then you have to do your transformations by hand
  - Override an existing method

Imagine a world with only transformations

- Then you can **break** your systems with just adding a method (as in VSCode :))

We need both refactorings and transformations

Example


There are 10 methods calling privateTransform
Select a strategy
Don't remove, but show me those senders
Remove, then browse senders
Remove it
Cancel
Accept

Code example

RBInlineMethodRefactoring >> preconditions

```
. . . .
self isOverridden ifTrue: [
 self refactoringWarning:
  ('<1p>>>><2s> is overridden. Do you want to inline it
  anyway?'
    expandMacrosWith: self classOfTheMethodToInline
    with: self inlineSelector) ] ].
. . . . .
```

Transformations AND Refactorings

- G. De Souza Santos defined Transformations and CompositeTransformations
- But a lot of code duplication
- Difficult to understand when using what
- What about preconditions?

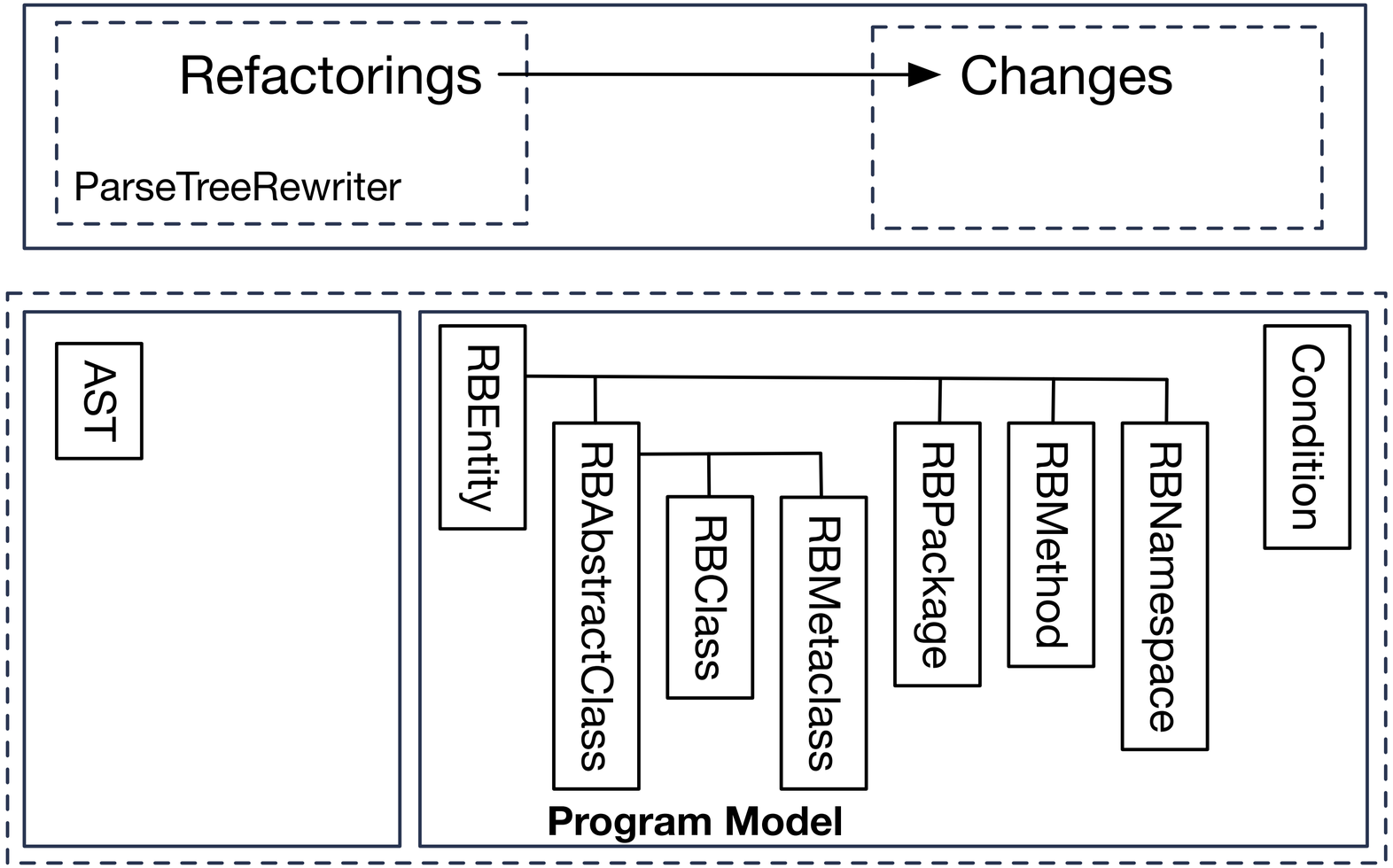
Goals of our engineering effort

- Modernise our engine
  - Reduce code duplication
  - Cleaner code
  - More tests
  - Assess refactorings (clear/correct preconditions / semantics)
  - Usability issues

Goals of our scientific work

- Reuse of transformations and refactorings to form new ones
- Understand composition issues (ongoing)
- Our ultimate goal is
  - Support you to write your own transformation
  - Domain specific refactoring definitions

A solid basis



- A refactoring reasons on a **program model**
- **Check preconditions** on such a program model
- Produces first class **changes** that can be **previewed**
- **Then and only then** actual modifications are done

A kind atomic approach

- A refactoring reasons on a program model
- Check preconditions on such a program model
- Produces first class changes that can be previewed
- **Then if something fails you can not apply the modifications**

About Preconditions

Do you think that transformations needs preconditions?

“Transformation-based Refactorings” [IWST22] shows that there are different kinds of preconditions

Transformation-based Refactorings: a First Analysis

N. Anquetil[1], M. Campero[1], Stéphane Ducasse[1], J.-P. Sandoval Alcocer[2] and P. Tesone[1]

[1]Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRIStAL, F-59000 Lille, France
[2]Pontificia Universidad Catolica de Chile, Santiago, Chile

Abstract
Refactorings are behavior preserving transformations. Little work exists on the analysis of their *implementation* and in particular how refactorings could be composed from smaller, reusable, parts (being simple transformations or other refactorings) and how (non behavior preserving) transformations could be used in isolation or to compose new refactoring operators. In this article we study the seminal implementation and evolution of Refactorings as proposed in the PhD of D. Roberts. Such an implementation is available as the Refactoring Browser package in Pharo. In particular we focus on the possibilities to reuse transformations independently from the behavior preserving aspect of a refactoring. The long term question we want to answer is: Is it possible to have more atomic transformations and refactorings composed out of such transformations? We study pre-conditions of existing refactorings and identify several families. We identify missed opportunities of reuse in the case of implicit composite refactorings. We analyze the refactorings that are explicitly composed out of other refactorings to understand whether the composition could be expressed at another level of abstraction. This analysis should be the basis for a more systematic expression of composable refactorings as well as the reuse of logic between transformations and refactorings.

1. Introduction

Refactorings are behavior preserving code transformations. The seminal work of Opdyke [Opd92] and the Refactorings Browser (first implementation of Refactorings of Roberts and Brant [RBJO96, RBJ97, BR98]) paved the way to the spread of refactorings [FBB+99]. They are now a must-have standard in modern IDEs [MHPB11, NCV+13, VCN+12, VCM+13, GDMH12]. A lot of research has been performed on refactorings such as for their detection [TME+18], missed application opportunities [TC09, TC10], practitioner use [MHPB11, VCN+12, NCV+13, VCM+13], or atomic refactorings for live environments [TPF+18]. Several publications focus on scripting refactorings [VEdM06, LT12, SvP12, HKV12, KBD15]. Finally, some work tried to speed up refactoring engines, proposing alternatives to the slow and bogus Java refactoring engine [KBDA16]. Related to this, it should be noted that the Pharo Refactoring Browser architecture supports fast pre-condition validation and refactoring execution and does not suffer from the architecture problems reported by Kim et al. [KBDA16].

IWST'22: International Conference of Smalltalk Technologies, August 24–26, 2022, Novy Sad, Serbia
nicolas.anquetil@inria.fr (N. Anquetil); stephane.ducasse@inria.fr (S. Ducasse); juanpablo.sandoval@ing.puc.cl (J.-P.S. Alcocer); pablo.tesone@inria.fr (P. Tesone)
0000-0002-5615-6691 (N. Anquetil); 0000-0001-6070-6599 (S. Ducasse); 0000-0002-8335-4351 (J.-P.S. Alcocer); 0000-0002-5615-6691 (P. Tesone)

CEUR Workshop Proceedings (CEUR-WS.org)

Different kind of Preconditions

See IWST 22

- Applicability
- Breaking change
- Skipping
- Others

applicabilityPreconditions

```
RBAddMethodTransformation >> applicabilityPreconditions

	. . . class should exist . . .

	. . . method should be parsable . . .
```

breakingChangesPrecondition

```
RBAddMethodRefactoring >> applicabilityPreconditions

    . . . method shouldn’t be overridden . . .
```

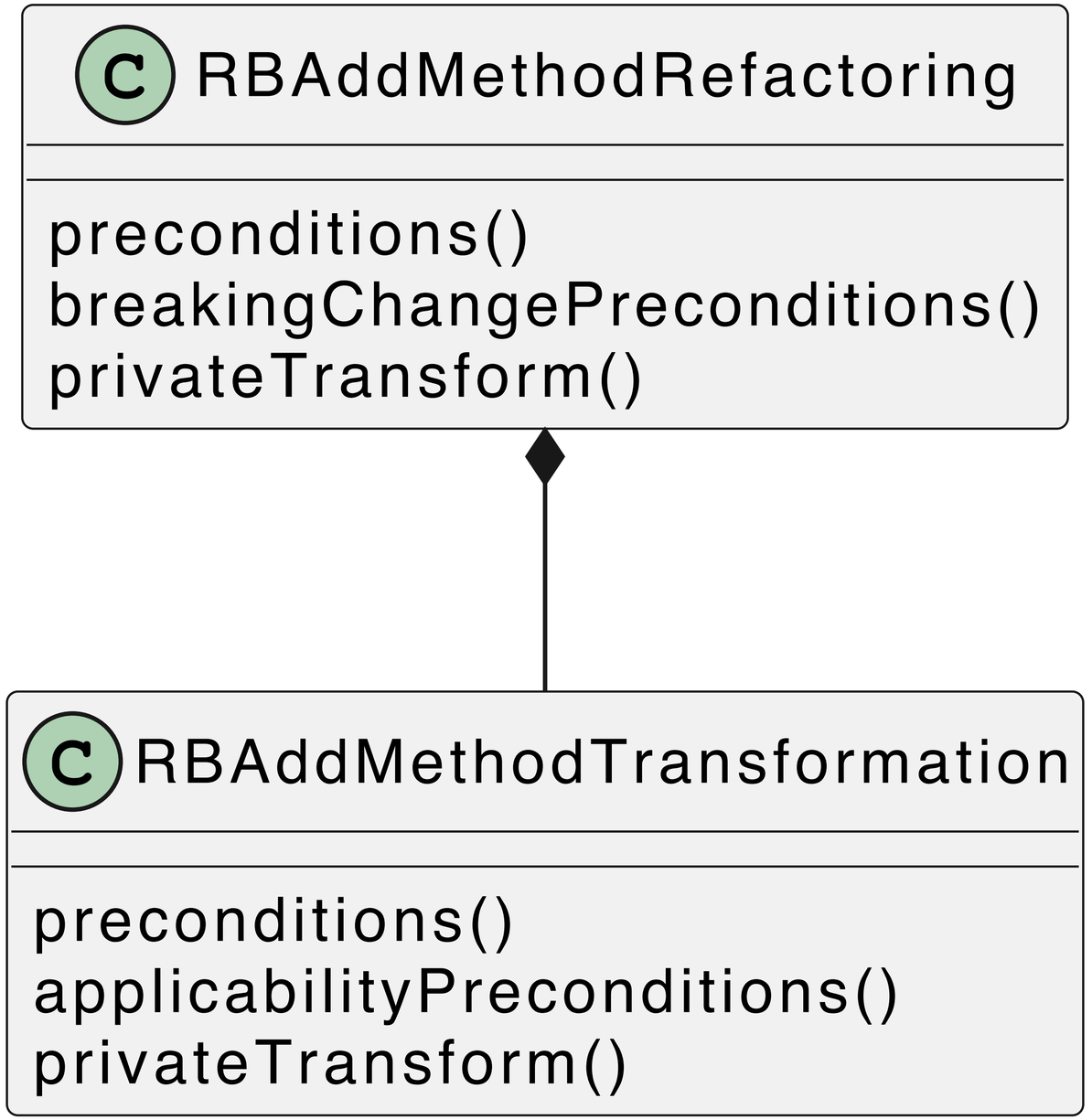
About Reuse

Clear separation

- Refactorings have breaking changes preconditions
- Transformations have applicability preconditions

Example

Add method



Example

Add method

```
RBAddMethodRefactoring >> preconditions

        transformation checkPreconditions.

        ^ self breakingChangePreconditions
```

```
RBAddMethodRefactoring >> privateTransform

        transformation privateTransform
```

```
RBAddMethodTransformation >> privateTransform

        self definingClass
                compile: sourceCode
                classified: protocol
```

Refactorings are decorators for transformations

RBAddMethodRefactoring —> RBAddMethodTransformation

- Refactoring uses Transformation to check applicability conditions
- Refactoring checks breaking change conditions
- Refactoring uses Transformation to make changes
- [Refactoring does cleanups and fixes if needed]

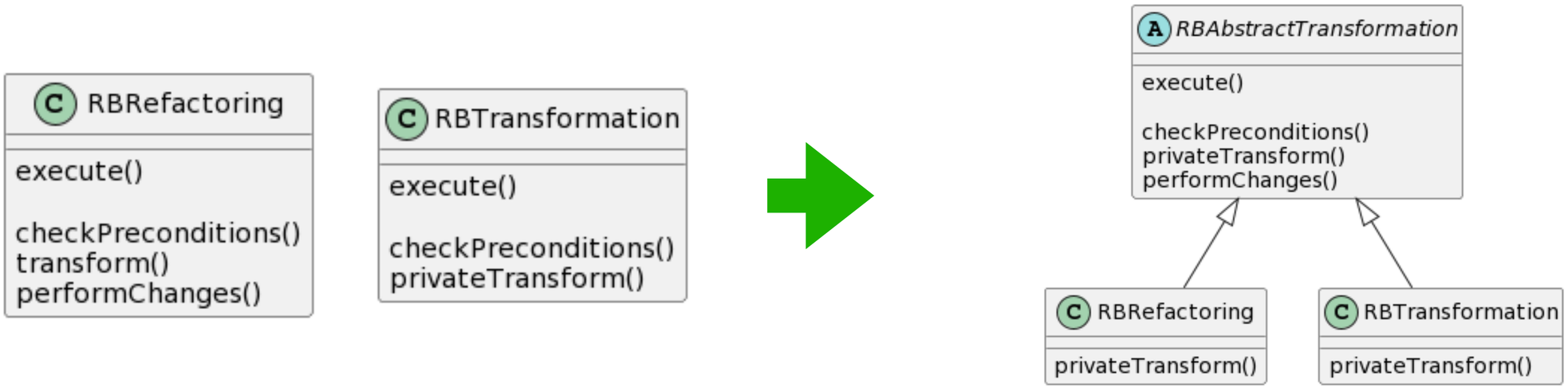
State of situation

- We are in the process of converting all the implementation to this design

About engineering

Realigning transformations and refactorings

- Better API
- Partial instantiation of refactorings to support better interaction
- Moving more responsibilities to refactorings

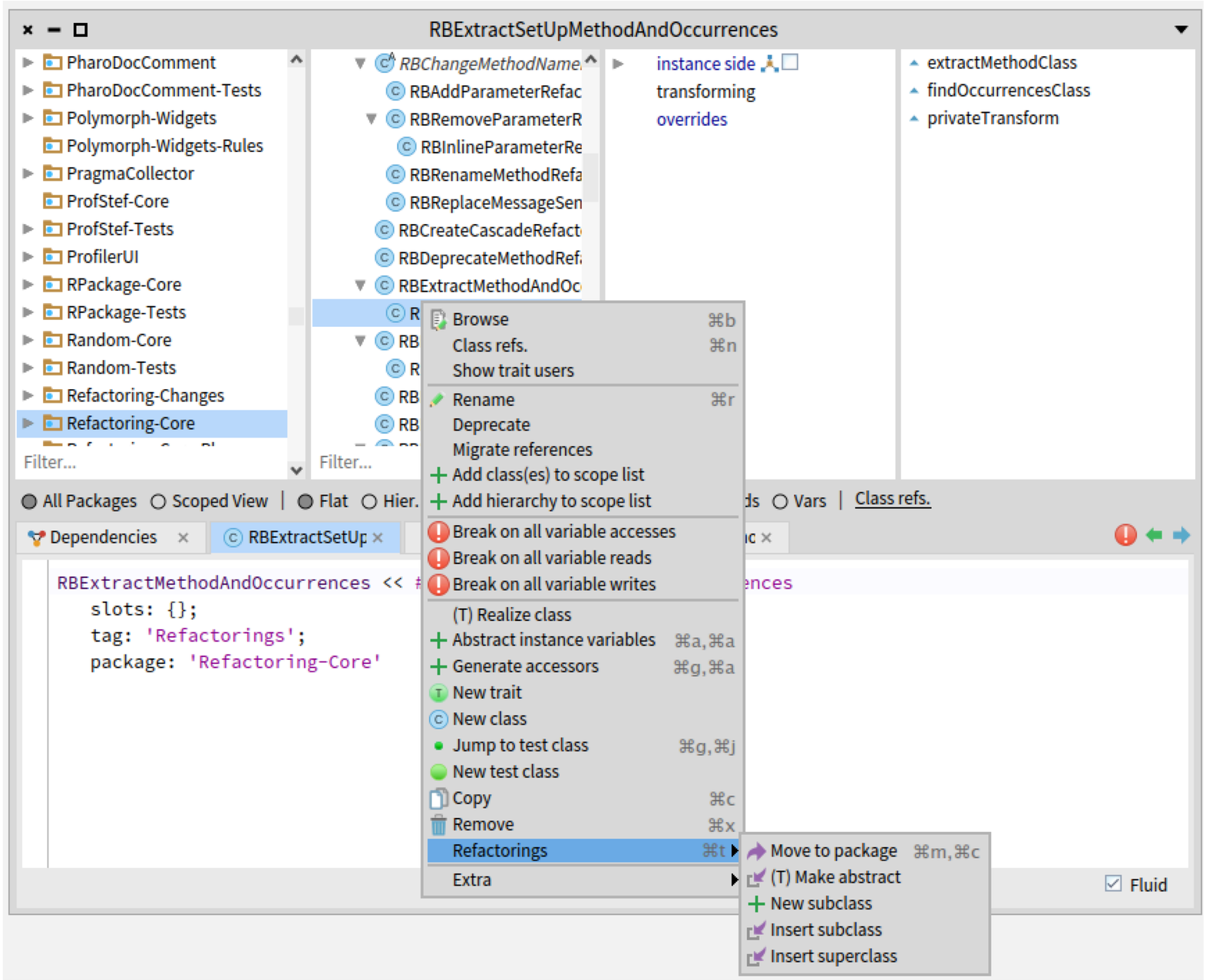


Revisit preconditions

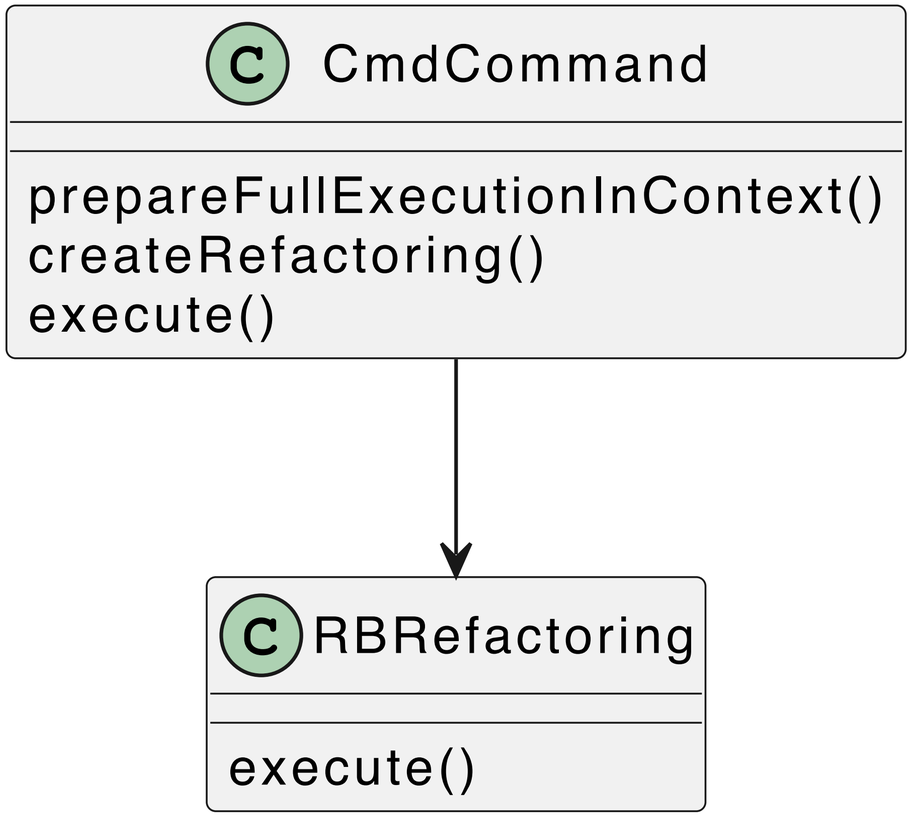
- Some preconditions were obscure / wrong
- Clearly identify breaking and applicability preconditions
- Adding a lot of comments
- Fixing, enhancing tests

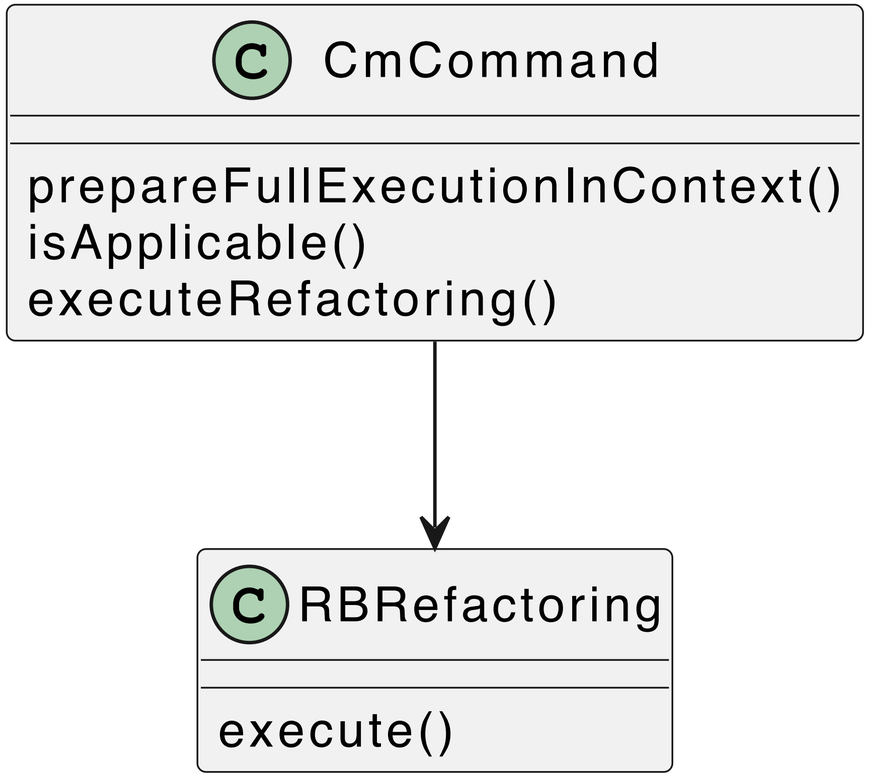
About the (T)

- You are warn when you use a Transformation

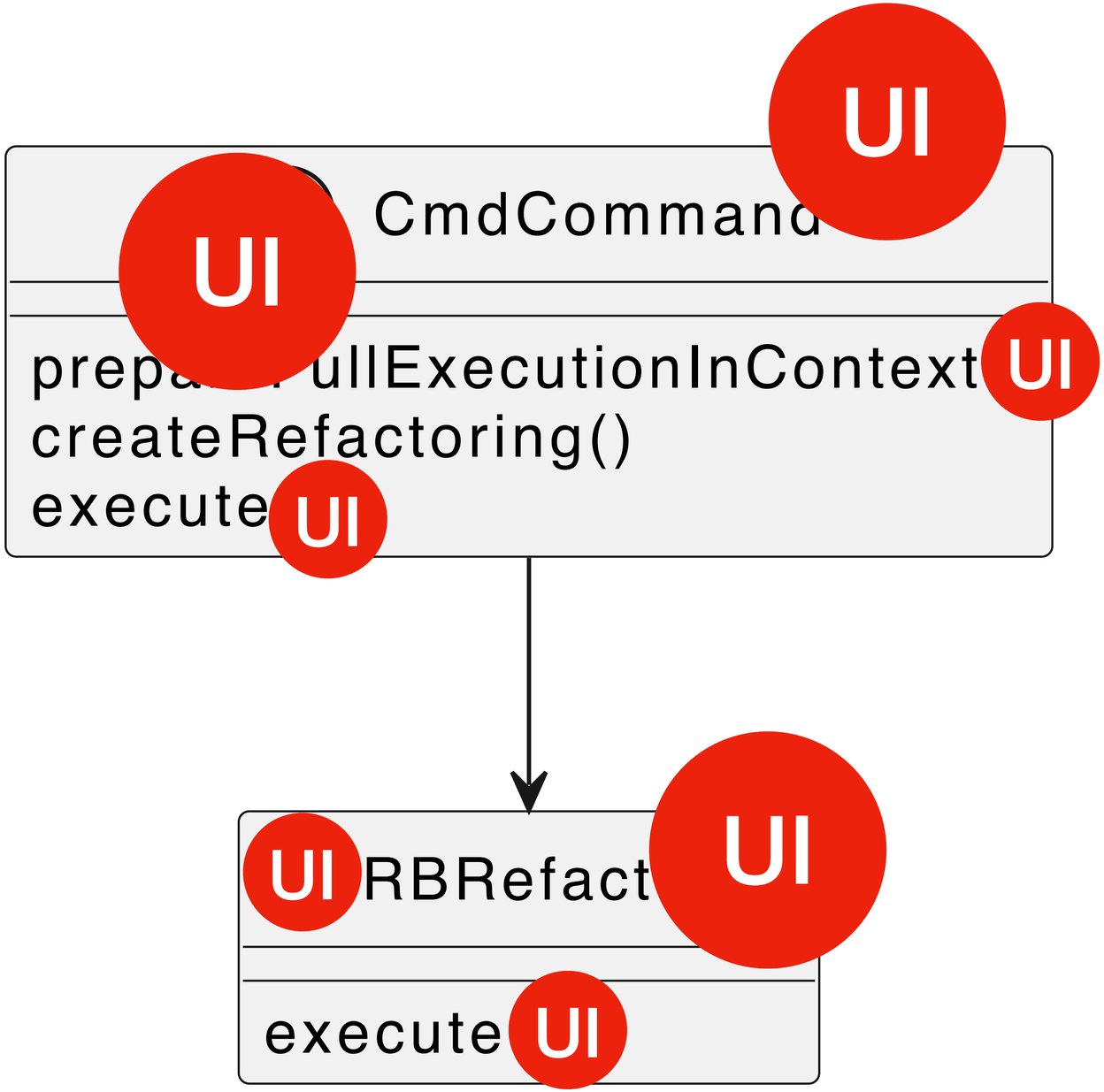


Cmd to Cm2.0


CmdCommand
prepareFullExecutionInContext()
createRefactoring()
execute()
RBRefactoring
execute()


CmCommand
prepareFullExecutionInContext()
isApplicable()
executeRefactoring()
RBRefactoring
execute()

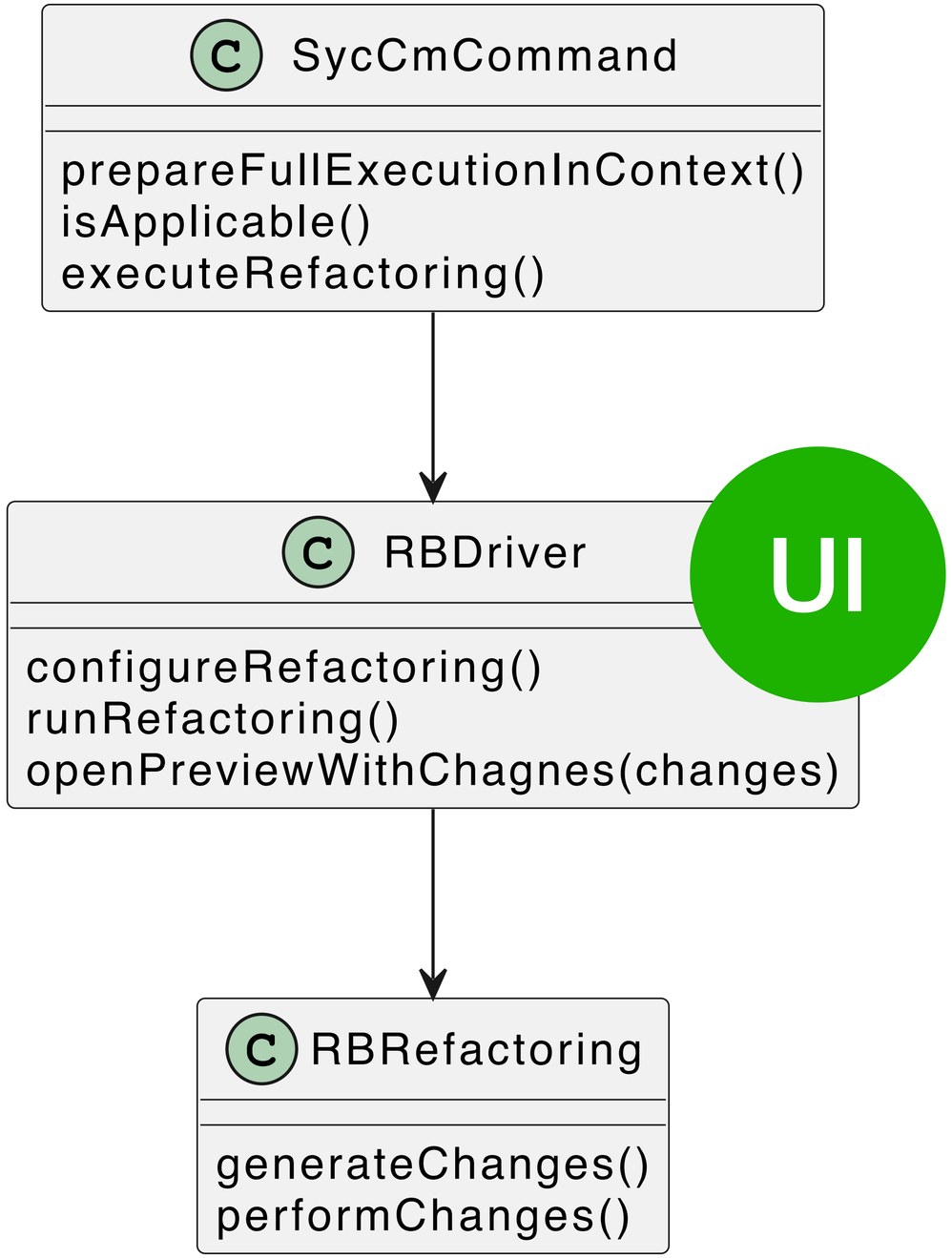
About UI


UI
UI
CmdCommand
prepa ullExecutionInContext
UI
createRefactoring()
execute
UI
UI
RBRefact
UI
execute
UI

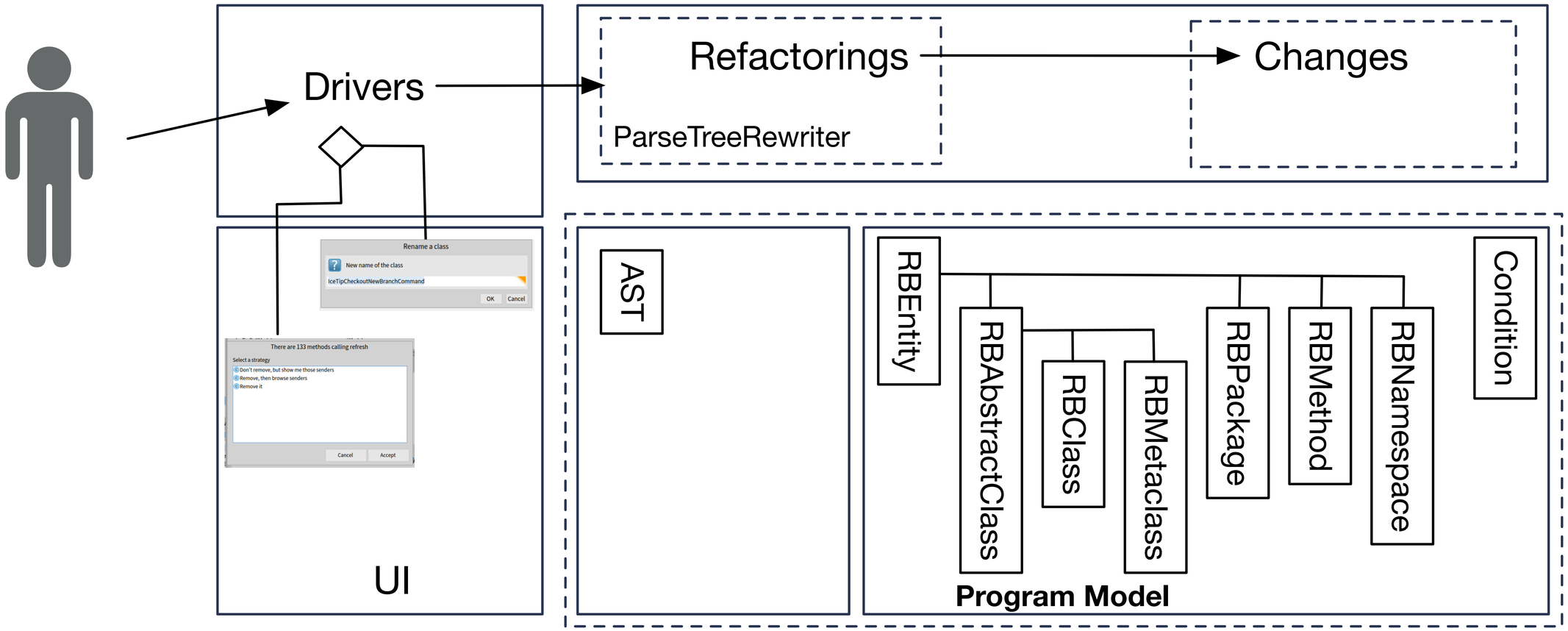
Preconditions should not raise UI!

- Preconditions had a lot of UI like:
  - Gather user input
  - Raise warnings
  - Show confirmation dialogs

New Tooling


SycCmCommand
prepareFullExecutionInContext()
isApplicable()
executeRefactoring()
RBDriver
UI
configureRefactoring()
runRefactoring()
openPreviewWithChagnes(changes)
RBRefactoring
generateChanges()
performChanges()

New Architecture



Driver

- UI is Driver’s responsibility now
  - Configures refactorings
  - Gathers user input
  - Displays errors and warnings
  - Displays any other relevant information (notifications, browsers, etc.)

Open questions

- Do we keep warning and exceptions
- Why not having failing reports that can be nicely displayed

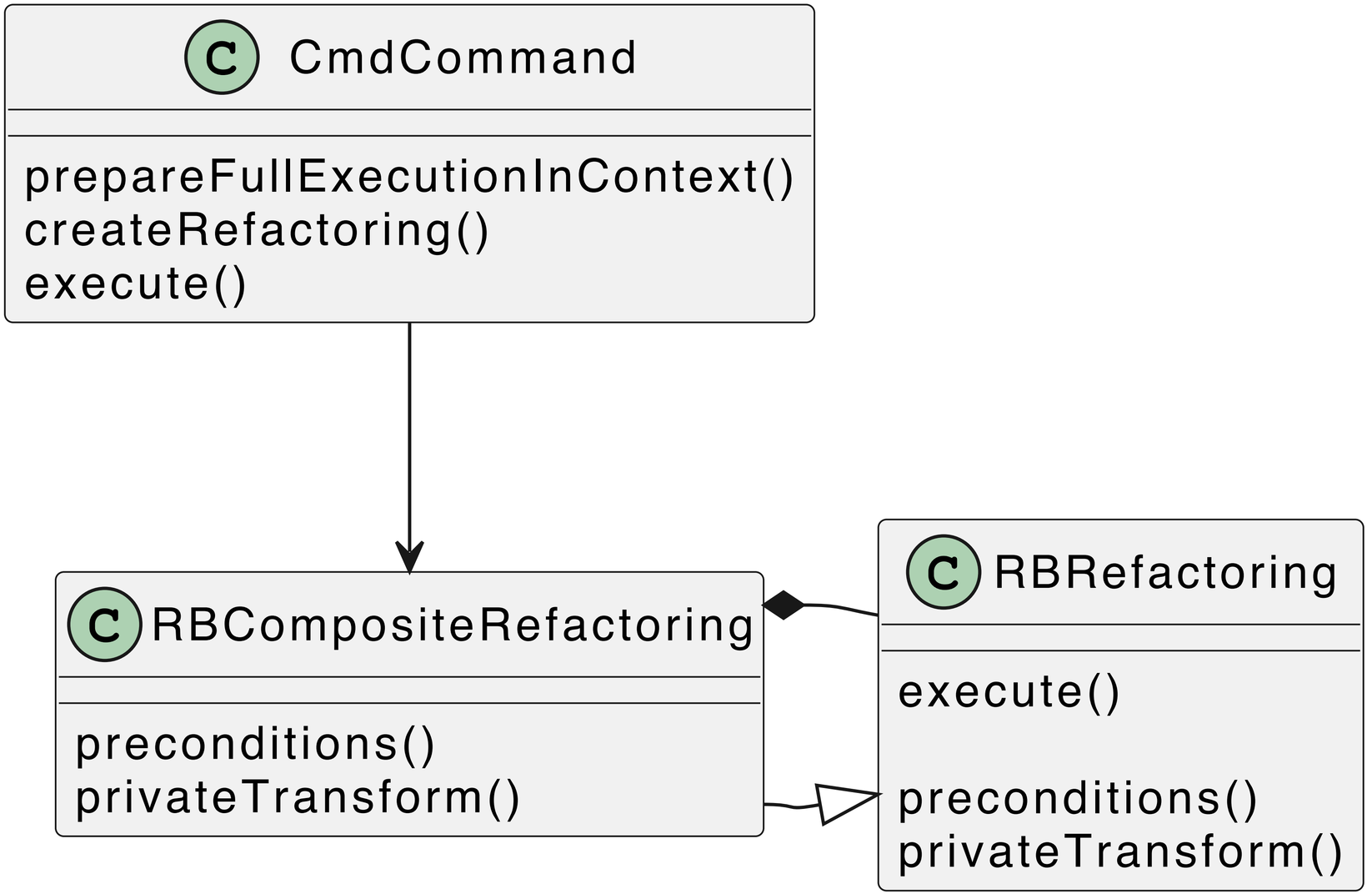
About Composition

(Started recently)

Existing Refactorings are monolithic

- G. De Souza Santos started to define more modular transformations (RBCompositeTransformation)
- We introduced
  - RBCompositeRefactoring
- Starting to play with composition semantics :)

Composite


CmdCommand
prepareFullExecutionInContext()
createRefactoring()
execute()
RBCompositeRefactoring
preconditions()
privateTransform()
RBRefactoring
execute()
preconditions()
privateTransform()

Let us study RemoveInstanceVariables

```
refactoring := RBCompositeRefactoring new
        model: model;
        refactorings: (variables collect: [:each |
                RBRemoveInstanceVariableRefactoring
                        model: model
                        remove: each
                        from: class]);
        yourself
```

RBCompositeRefactoring

- Execute in sequence refactorings
- P1, T1; P2, T2; … Pn Tn

```
RBCompositeRefactoring >> privateTransform

        refactorings do: [ :each | each generateChanges ]
```

Different execution semantics

- Stop on failure (as RBCompositeRefactoring)
- Skip failed and proceed (as RBCompositeContinuingRefactoring)

```
RBCompositeContinuingRefactoring >> privateTransform

    refactorings do:
       [ :each |

          [ each generateChanges] on: RBRefactoringError do: [ :ex | ]

       ]
```

Custom composite example

Can we remove both fooUnik and barUnikUnik?

```
X >> fooUnik
 ^ 12

X >> barUnikUnik
      ^ self fooUnik + 1
```

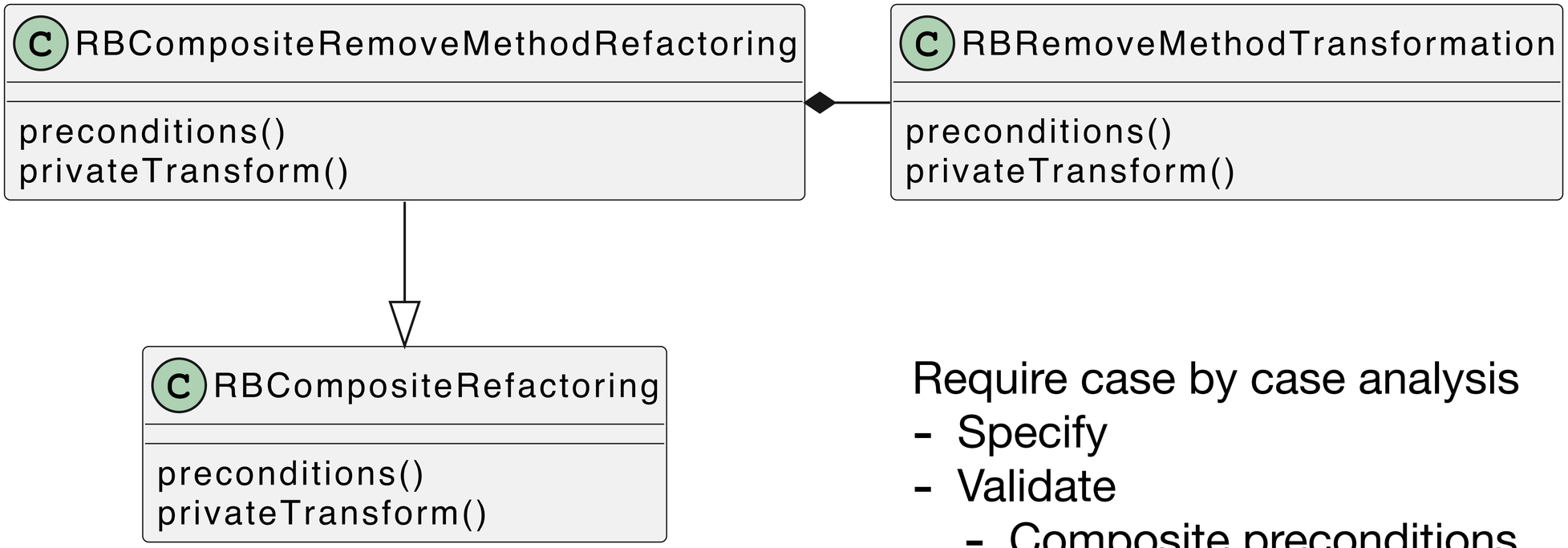
Custom composite need

removeMethods (fooUnik, barUnikUnik)

is not equals to

removeMethod (fooUnik);
removeMethod (barUnikUnik)

Custom composite



Require case by case analysis
- Specify
- Validate
  - Composite preconditions
  - Component preconditions

Future work: a large effort

- Continue eliminating code duplication between refactorings and transformation
- Leverage composition of refactorings where possible
- Migrate to Commander2.0
- Migrate all UI to Driver
- A lot more to…

New architecture for the future :)

- Many many hidden improvements
- Driver for interactive application
- Clear roles for Transformations and Refactoring:
  - A refactoring is a decorator of a transformation
  - Better separation of concerns

Still some work but the path is clear