



HAL
open science

An extensible production-level debugger

Adrien Vanègue, Steven Costiou

► **To cite this version:**

Adrien Vanègue, Steven Costiou. An extensible production-level debugger. Journées Nationales du GDR-GPL 2023, Jun 2023, Rennes, France. hal-04130163

HAL Id: hal-04130163

<https://inria.hal.science/hal-04130163>

Submitted on 15 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

An extensible production-level debugger

Adrien Vanègue

Steven Costiou

Problem: building and evaluating new debuggers

Debuggers are difficult to build.

- Huge development costs
- Difficult to reuse: prototypes are often one-shot
- Costly and difficult to maintain
- Difficult to test

How to make debuggers extensible to facilitate the construction and maintenance of new prototypes?

Debuggers are difficult to evaluate.

- Prototypes are often unstable
- Difficult to evaluate on real-world scenarios: practitioners need to adopt and use the new debugger

How to make debuggers solid to make them usable in practice and evaluate them?

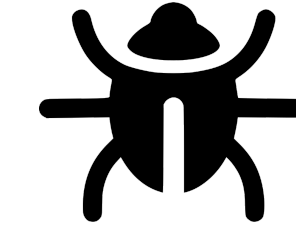
Requirements for extensible production-level debugger

Modular and easy to reuse, test and maintain

extensible and easy to build on top

production-level and regularly maintained

Testing

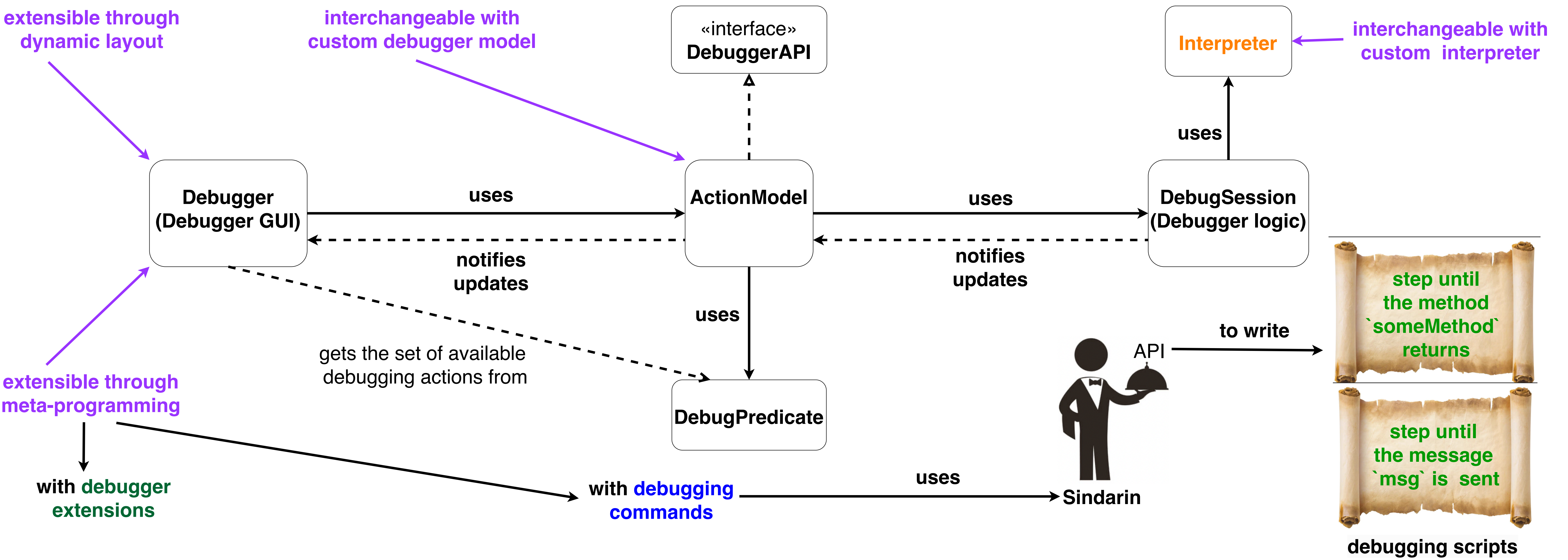


Tools resulting from prototypes are stable, through testing and bug fixes



Easy to adopt: practitioners use the debugger they know along with the new tool

The Pharo debugger design



Building new debugging tools: Examples

CHEST:

a global collection with an API to access objects from everywhere

- copy `a` from program 1
- paste `a` from program 1 into program 2

while debugging code `methodWithTwoAssignments`

```

a := 1.
a := 5.
^ Point x: 5 y: '3' asInteger
    
```

store `a` in chest

```

inChest: 'ClipboardDefault' at: 'clipboardEntry' put: a
    
```

API Example

load `a` from program 1 into program 2

```

toto := 42.
tata := (Chest named: 'ClipboardDefault') at: 'clipboardEntry'
    
```

How to compare my objects between program 1 and 2?

```

toto == tata
false
    
```

inside program 2

integrated as debugger extension

JUMP TO CARET:

a Goto command to debug

Scenario: we expect our code to take the `ifFalse:` path but takes the `ifTrue:` path

```

a < 100
ifTrue: [ 'doSomeCode' printString ]
ifFalse: [ 'doSomeOtherCode' printString ]
    
```

we are here

we want to go there, to see what would happen

We can jump to caret, to see what would happen if the other branch was executed

The program state is identical before and after having jumped to caret! We can now debug this branch!

```

a < 100
ifTrue: [ 'doSomeCode' printString ]
ifFalse: [ 'doSomeOtherCode' printString ]
    
```

we are now there

A production debugger integrated into the Pharo language

THALES

REAL USERS

- moldable debugger, possible to add:
 - specific debugging operations
 - specific debugging tools
- new debugging tools can be used as extensions of the existing debugger
- Concerns are separated as each component is independent: the interpreter, the model, the API, the UI, the debugging commands, and the debugger extensions
- 3 main contributors, 958 commits
- Listening to feedback from users
- The debugger is used for empirical evaluation of research prototypes
- Lots of contributions merged into the Pharo baseline
- 559 tests