Amélioration du design et des fonctionnalités du debugger Pharo

Stage de Adrien Vanègue,
tutoré par:

M. S. Costiou

Inria

Université de Lille

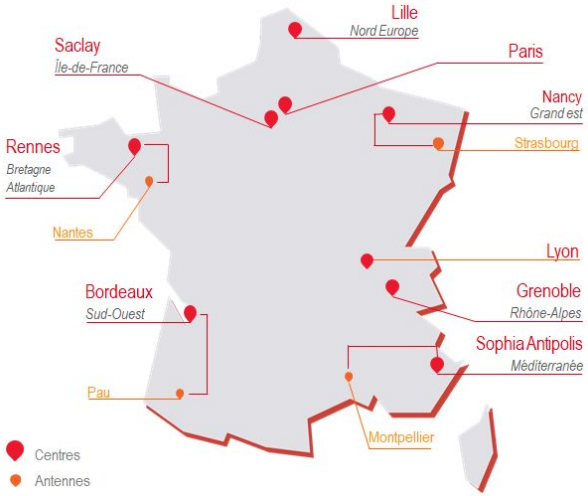
M. R. Rouvoy

Présentation d'Inria

200 équipes-projets

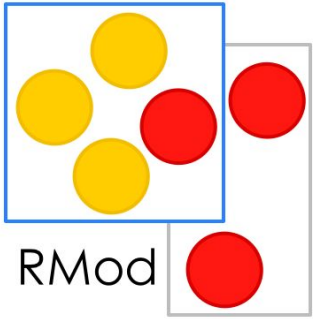
INVENTEURS DU MONDE NUMÉRIQUE

3900 chercheurs



Présentation de l’équipe RMOD

évolution de software



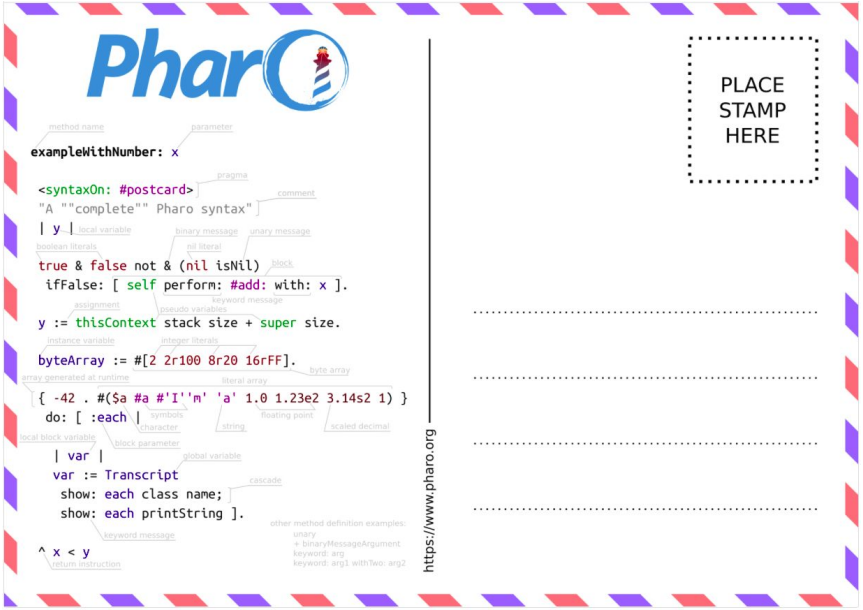
amélioration d’outils
(debuggers, VM…)

création de nouveaux outils

Présentation de Pharo

réflexif

tout est objet



Contexte du stage

Object Centric debugging REloaded

vise à

- construire des prototypes de debuggers objet-centriques exportables vers l'open-source
- évaluer les bénéfices de l'object-centric debugging

Projet-ANR-21-CE25-0004

nécessite

infrastructure solide

Objectif du stage

Le debugger Pharo

pile d'appel de méthodes

barre d'outils (steps…)

code

inspecteur

ZeroDivide

Stack

Class	Method	Package
SmallInteger	/	Kernel
UndefinedObject	DoIt	-
OpalCompiler	[receiver withArgs: (context ifNil: [#()] ifNot	OpalCompiler-Core
FullBlockClosure (BlockClosure)	on:do:	Kernel

Proceed | Into | Over | Through | Run to | Restart | Return | Where is? | Create | Advanced Step

Next instance creation | Next call in receiver | [Exp.] Skip through (stop after) | To method entry | To return | Next call in class | [Exp.] Skip up to (stop before)

```
5   No Look
6
7   <primitive: 10>
8   aNumber = 0 ifTrue: [1 + 2. ^(ZeroDivide dividend: self) signal].
9
10  "This ugly type checking is needed because the implementation in the superclass
    assumes that the argument is a LargeInteger.
11  So, it is using a primitive message that is only in LargeInteger.
12  Using the implementation in the superclass when we have a SmallInteger will fail
    the primitive and use the slow code.
13  At the end it is creating the same Fraction that is done here.
14  These duplication has to be avoided by correctly implementing the decision by
    using double dispatch."
15
16  ^(aNumber isMemberOf: SmallInteger)
```

a SmallInteger (1)

Type	Variable	Value
implicit	Σ self	1
arg	Σ aNumber	0
implicit	Σ stackTop	0
implicit	thisContext	SmallInteger>>/
implicit	Exception	ZeroDivide

commandes de stepping avancées (= **Sindarin**)

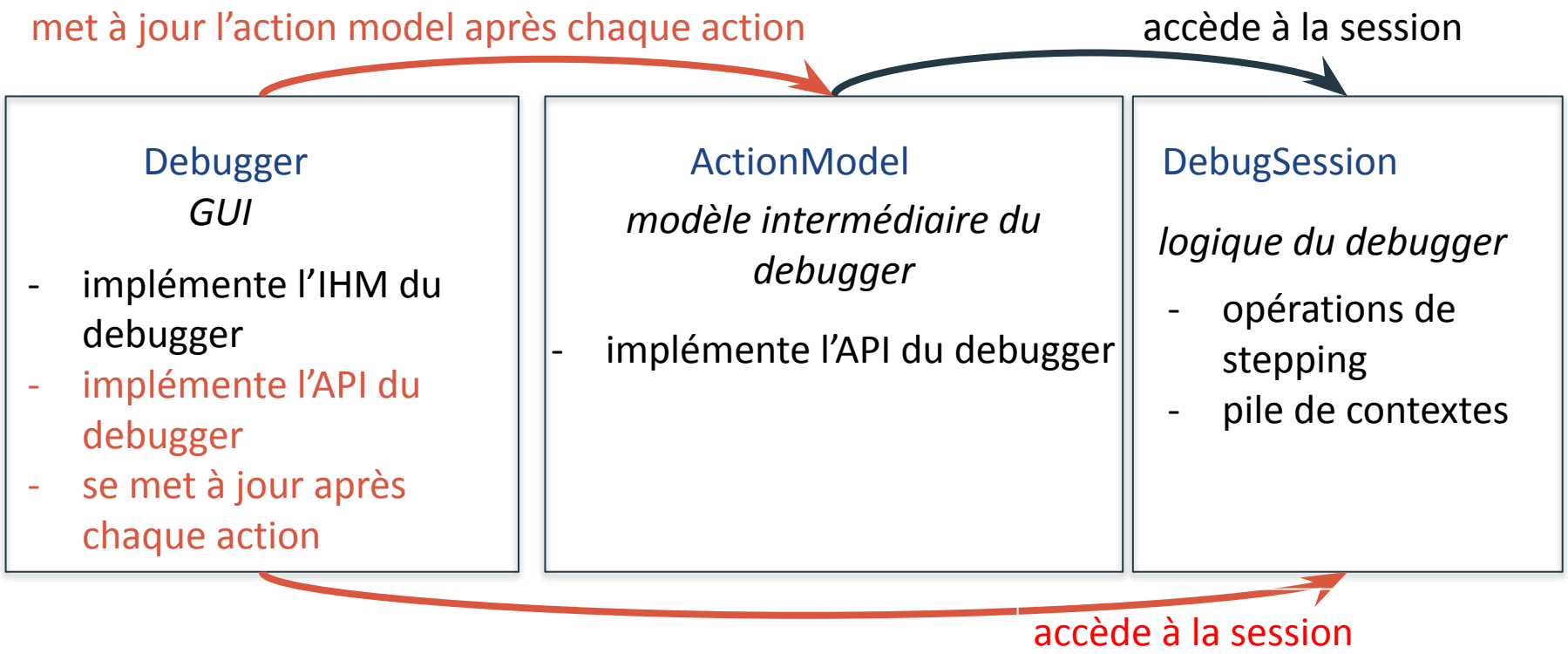
Mes contributions / **plan**

11 PR dont 8 intégrées et 3 en review sur le debugger Pharo

code testé unitairement

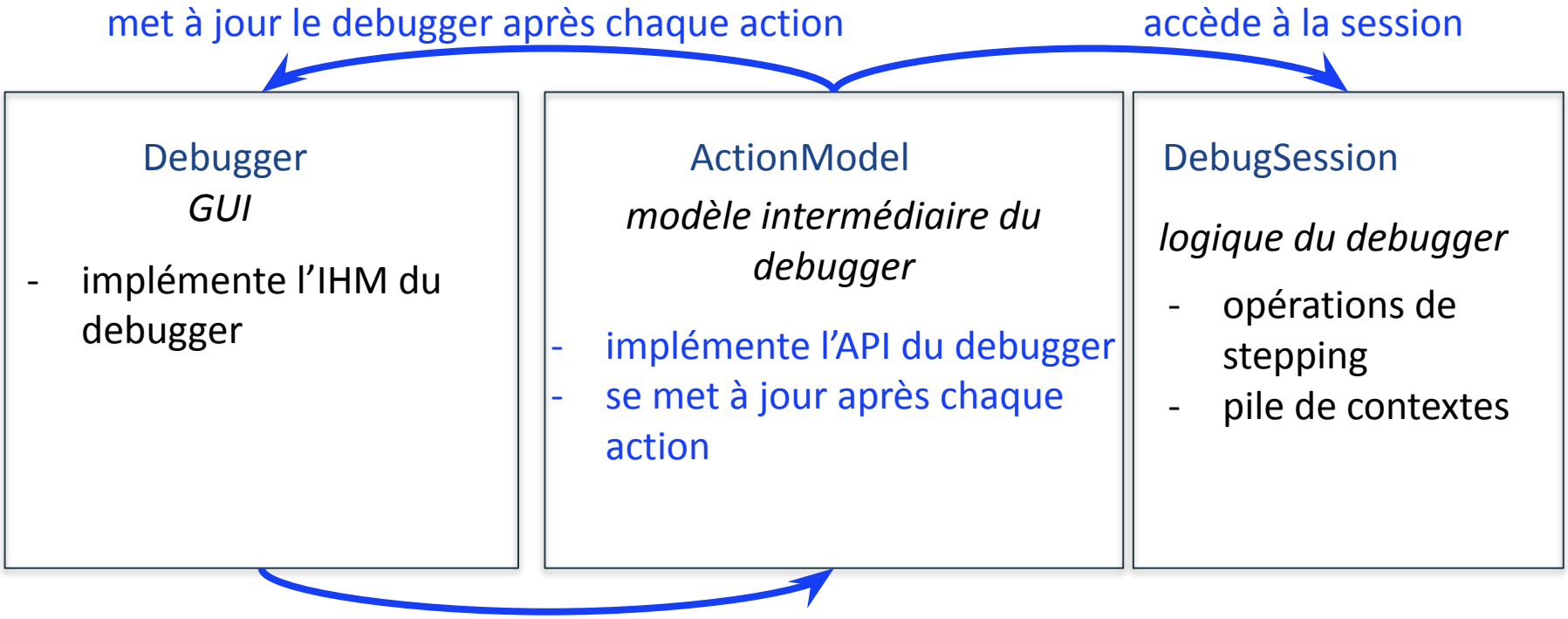
- **Séparation IHM / API / modèle**
- Correction de commandes de debugging avancées (Sindarin)
  - **skip**
  - **stepToReturn**
- Outillage par Chest, un gestionnaire de coffres-forts
- **Participation à la rédaction d'un papier de recherche**
- Mise en forme dynamique pour afficher dynamiquement les plugins

Séparation des préoccupations



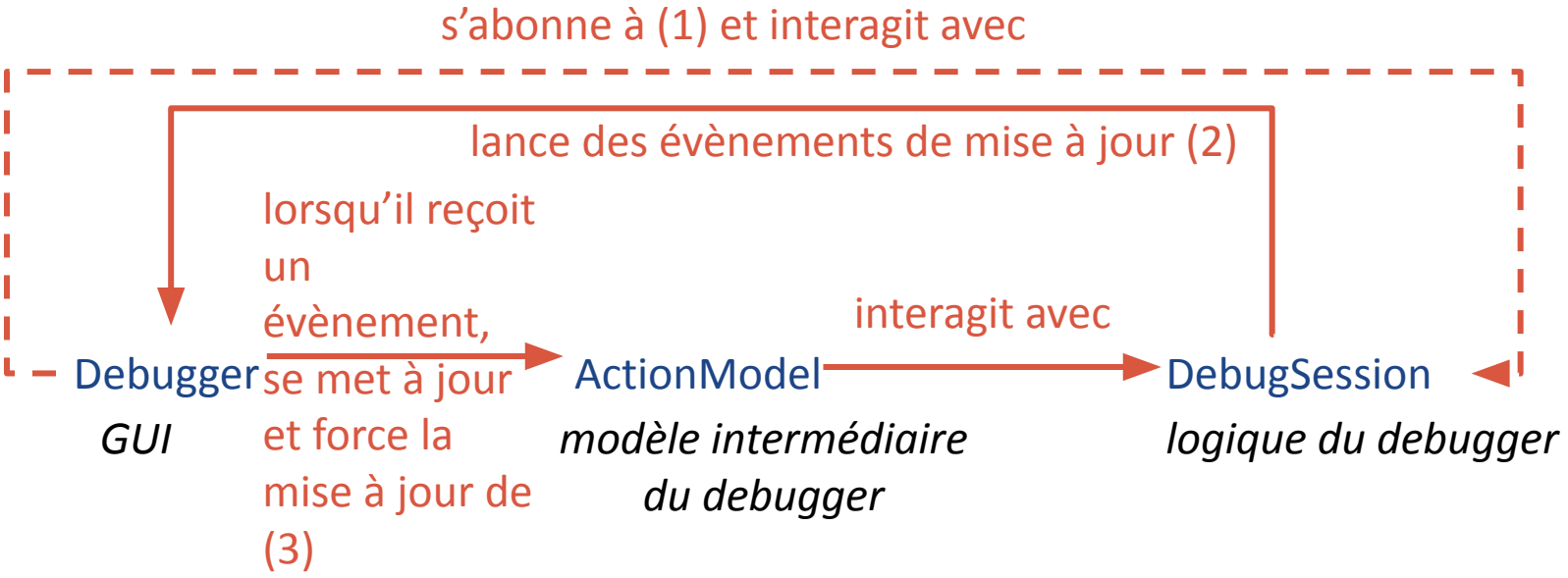
Légende: avant

Séparation des préoccupations



Légende: après

Séparation des préoccupations : refonte du système d'évènements

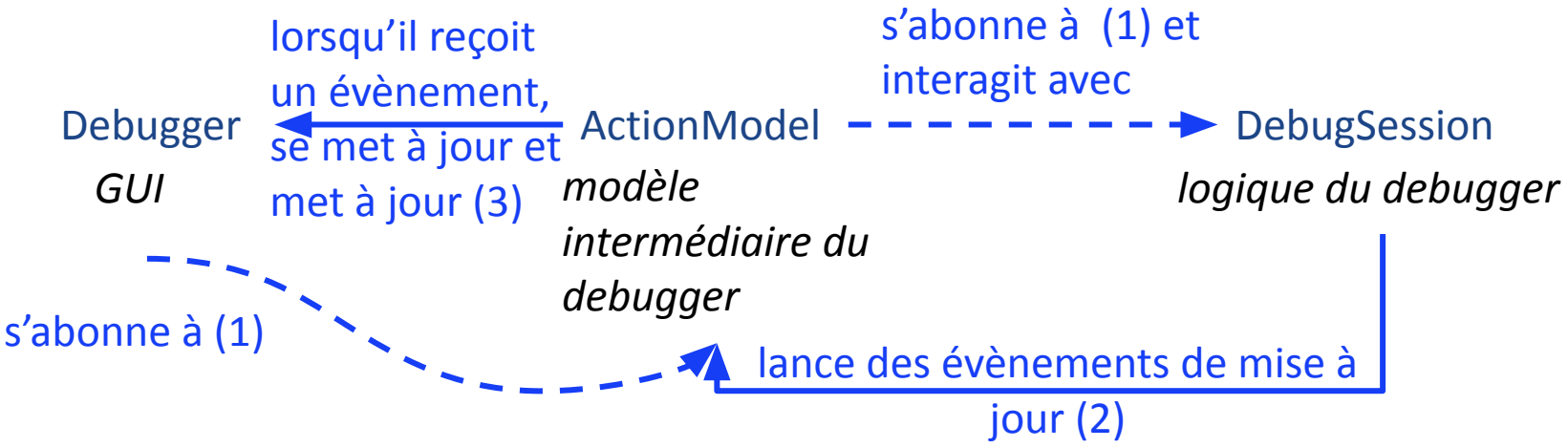


Légende: avant

abonnement

(1): 1ère action impliquée dans une succession d'évènements

Séparation des préoccupations : refonte du système d'évènements



Légende: après

abonnement

(1): 1ère action impliquée dans une succession d'évènements

Sindarin : la commande skip, comportement attendu

UndefinedObject>>DoIt

Stack

Class	Metho
UndefinedObject	DoIt
CompiledMethod	valueW
SpCodeDebugItCommand	[aCor

Proceed Into Over Through Ru

```
1 DoIt
2
3     | a b |
4     a := 1.
5     a := b := 3 + 2.
6     ^ 42
```

1	nil
2	nil
3	1

Bytecode

```
49 <51> pushConstant: 1
50 <D0> popIntoTemp: 0
51 <20> pushConstant: 3
52 <21> pushConstant: 2
53 <60> send: +
54 <F5 01> storeIntoTemp: 1
56 <D0> popIntoTemp: 0
57 <22> pushConstant: 42
58 <5C> returnTop
```

Variable	Va
stackTop	1
a	nil
b	nil

UndefinedObject>>DoIt

Stack

Class	Metho
UndefinedObject	DoIt
CompiledMethod	valueW
SpCodeDebugItCommand	[aCor

Proceed Into Over Through Rur

```
1 DoIt
2
3     | a b |
4     a := 1.
5     a := b := 3 + 2.
6     ^ 42
```

1	nil
2	nil
3	3
4	2

Bytecode

```
49 <51> pushConstant: 1
50 <D0> popIntoTemp: 0
51 <20> pushConstant: 3
52 <21> pushConstant: 2
53 <60> send: +
54 <F5 01> storeIntoTemp: 1
56 <D0> popIntoTemp: 0
57 <22> pushConstant: 42
58 <5C> returnTop
```

Variable	V
stackTop	2
a	nil
b	nil

Sindarin : la commande skip, comportement réel

UndefinedObject>>DoIt

stepOver

Stack

Class	Method
UndefinedObject	DoIt
CompiledMethod	valueW
SpCodeDebugItCommand	[aCon

Proceed Into Over Through Run

```
1 DoIt
2
3     | a b |
4     a := 1.
5     a := b := 3 + 2.
6     ^ 42
```

1	nil
2	nil
3	nil
4	2

Bytecode

```
49 <51> pushConstant: 1
50 <D0> popIntoTemp: 0
51 <20> pushConstant: 3
52 <21> pushConstant: 2
53 <60> send: +
54 <F5 01> storeIntoTemp: 1
56 <D0> popIntoTemp: 0
57 <22> pushConstant: 42
58 <5C> returnTop
```

Variable	
stackTop	2
a	nil
b	nil

Instance of nil did not understand #+

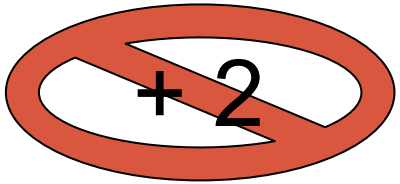
Stack

Class	Method	Package
MessageNotUnderstood (Exception)	signal	Kernel
UndefinedObject (Object)	doesNotUnderstand:	Kernel
UndefinedObject	doesNotUnderstand:	Kernel

Proceed Into Over Through Run to Restart Return Where is? Create Advanced Step

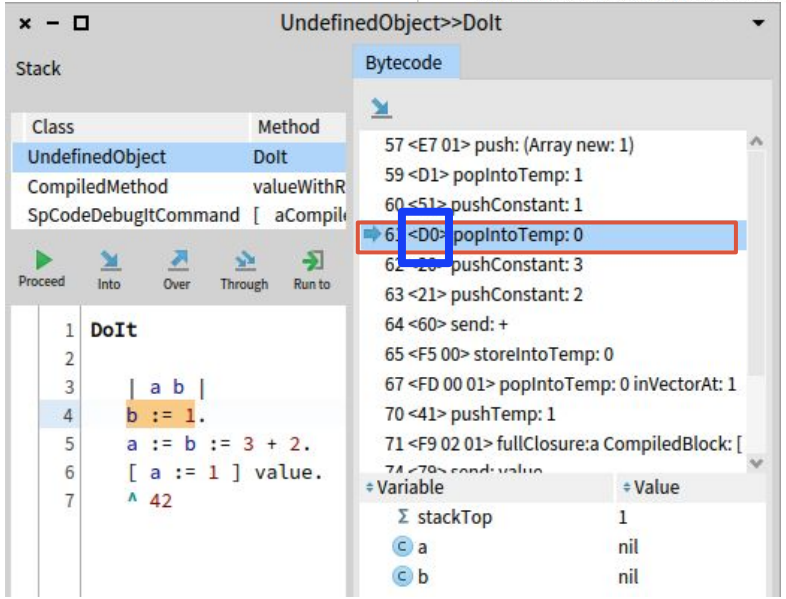
```
1 signal
2     "Ask ContextHandlers in the sender chain to handle th
  default is to execute and return my defaultAction."
3     <debuggerCompleteToSender>
4     signalContext := thisContext contextTag.
5     signaler ifNil: [ signaler := self receiver ].
6     ^ signalContext nextHandlerContext handleSignal: self
```

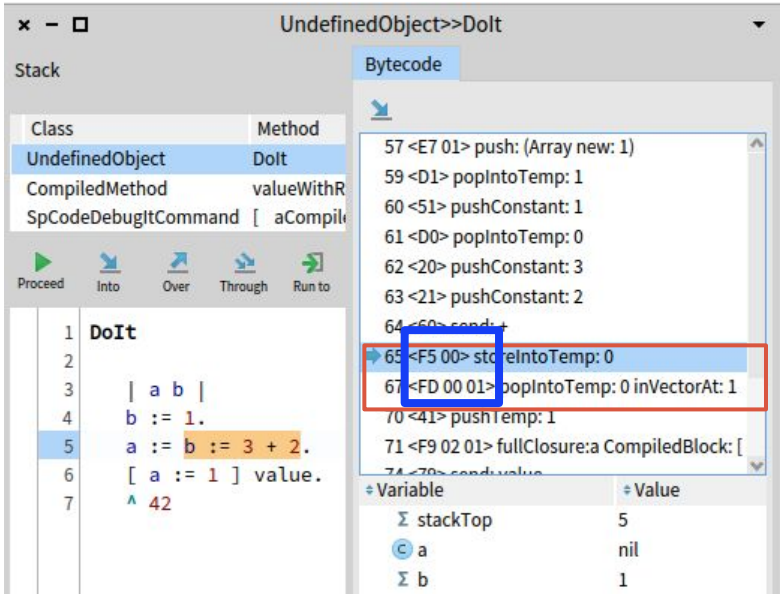
Sindarin : la commande skip, saut d'octets arbitraires




```
skipAssignmentNodeWith: replacementValue
   self context pop.
   "Pop the value to be assigned"
   "Push the replacement value on the context's value stack, to simulate that the assignment happened
and had value nil"
   self context push: replacementValue.
   "Increase the pc to go over the assignment"
   self context pc: self context pc + 2.
   "Execute bytecodes the debugger usually executes without stopping the execution (for example popping
the return value of the just executed message send if it is not used afterwards)"
   self debugSession
      stepToFirstInterestingBytecodeIn: self debugSession interruptedProcess
```

+ self currentBytecode bytes size





Sindarin : la commande skip, storeInto VS popInto

storeInto ifTrue:

```
skipAssignmentNodeWith: replacementValue
    self context pop.
    "Pop the value to be assigned"
    "Push the replacement value on the context's value stack, to simulate that the assignment happened
and had value nil"
    self context push: replacementValue.
    "Increase the pc to go over the assignment"
    self context pc: self context pc + 2.
    "Execute bytecodes the debugger usually executes without stopping the execution (for example popping
the return value of the just executed message send if it is not used afterwards)"
    self debugSession
        stepToFirstInterestingBytecodeIn: self debugSession interruptedProcess
```

popInto

UndefinedObject>>DoIt

Stack

Class	Method
UndefinedObject	DoIt
CompiledMethod	valueW

Proceed Into Over Through Run

```
1 DoIt
2
3     | a b |
4     a := 1.
5     a := b := 3 + 2.
6     ^ 42
```

Σ 1	1
2	nil
Σ 3	5

Bytecode

49 <51> pushConstant: 1
50 <D0> popIntoTemp: 0
51 <20> pushConstant: 3
52 <21> pushConstant: 2
53 <60> send: +
54 <F5 01> storeIntoTemp: 1
56 <D0> popIntoTemp: 0
57 <22> pushConstant: 42
58 <5C> returnTop

Variable	
Σ stackTop	5
Σ a	1
b	nil

UndefinedObject>>DoIt

Stack

Class	Method
UndefinedObject	DoIt
CompiledMethod	valueW

Proceed Into Over Through Run

```
1 DoIt
2
3     | a b |
4     a := 1.
5     a := b := 3 + 2.
```

Σ 1	1
Σ 2	5
Σ 3	5

Bytecode

49 <51> pushConstant: 1
50 <D0> popIntoTemp: 0
51 <20> pushConstant: 3
52 <21> pushConstant: 2
53 <60> send: +
54 <F5 01> storeIntoTemp: 1
56 <D0> popIntoTemp: 0
57 <22> pushConstant: 42
58 <5C> returnTop

Variable	
Σ stackTop	5
Σ a	
Σ b	5

UndefinedObject>>DoIt

Stack

Class	Metho
UndefinedObject	DoIt
CompiledMethod	valueW

Proceed Into Over Through Ru

```
1 DoIt
2
3     | a b |
4     a := 1.
5     a := b := 3 + 2.
6     ^ 42
```

Σ 1	5
Σ 2	5

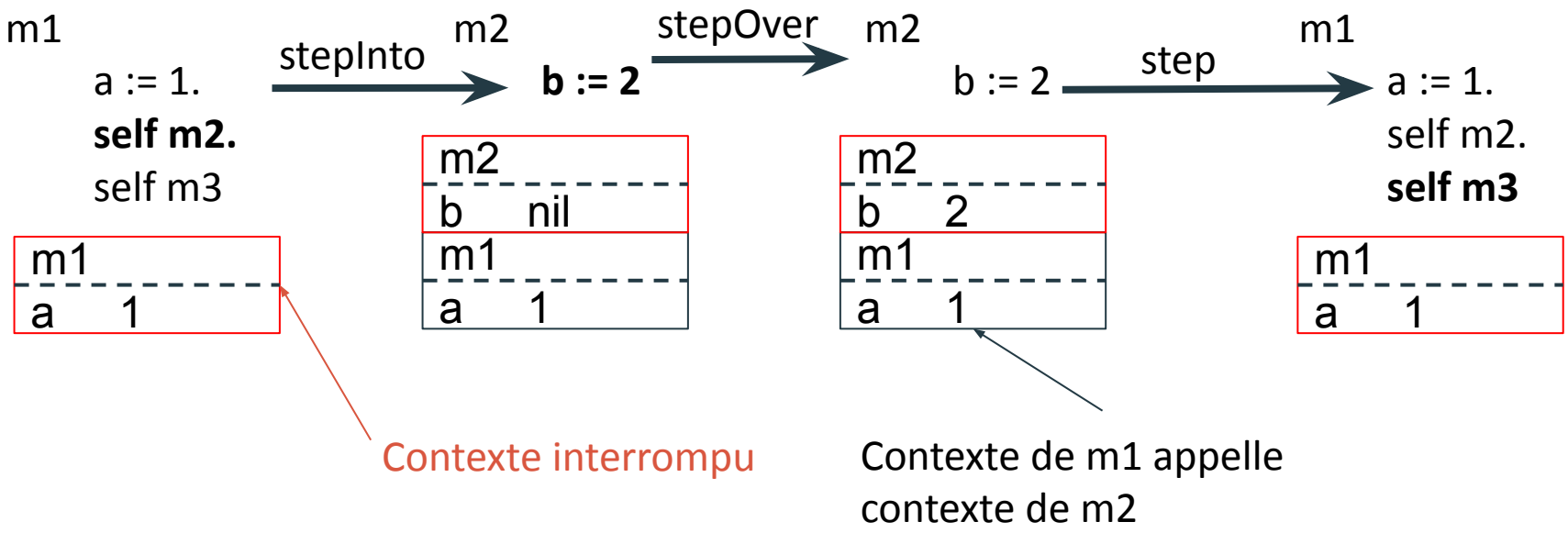
Bytecode

49 <51> pushConstant: 1
50 <D0> popIntoTemp: 0
51 <20> pushConstant: 3
52 <21> pushConstant: 2
53 <60> send: +
54 <F5 01> storeIntoTemp: 1
56 <D0> popIntoTemp: 0
57 <22> pushConstant: 42
58 <5C> returnTop

Variable	
Σ stackTop	5
Σ a	5
Σ b	5

Sindarin : la commande stepToReturn, mécanisme de step

Contexte = objet contenant toutes les informations dynamiques de l'exécution d'une méthode / fonction anonyme



Sindarin : la commande stepToReturn

comportement attendu

Stack

Class	Method
SindarinDebuggerTest	[^ 42]
SindarinDebuggerTest	helperMetl

Proceed Into Over Through Run to

```
1 helperMethodNonLocalR
2     | block |
3     block := [ ^ 42 ].
4     block value.
5     ^ 43
```

Stack

Class	Method
SindarinDebuggerTest	helperMethodNonL
UndefinedObject	DoIt

Proceed Into Over Through Run to Restart

```
1 helperMethodNonLocalReturn
2     | block |
3     block := [ ^ 42 ].
4     block value.
5     ^ 43
```

bloc = fonction anonyme
sauf que :
return dans un bloc
return dans la méthode
qui le définit

comportement réel

Stack

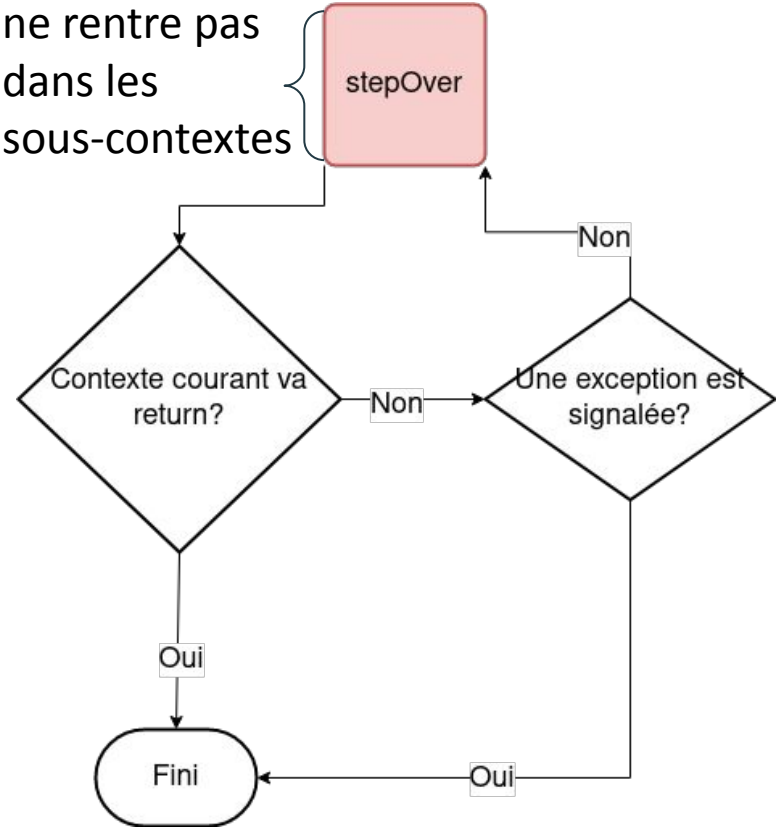
Class	Method	Pa
UndefinedObject	DoIt	-
CompiledMethod	valueWithReceiver:argumer	Ker

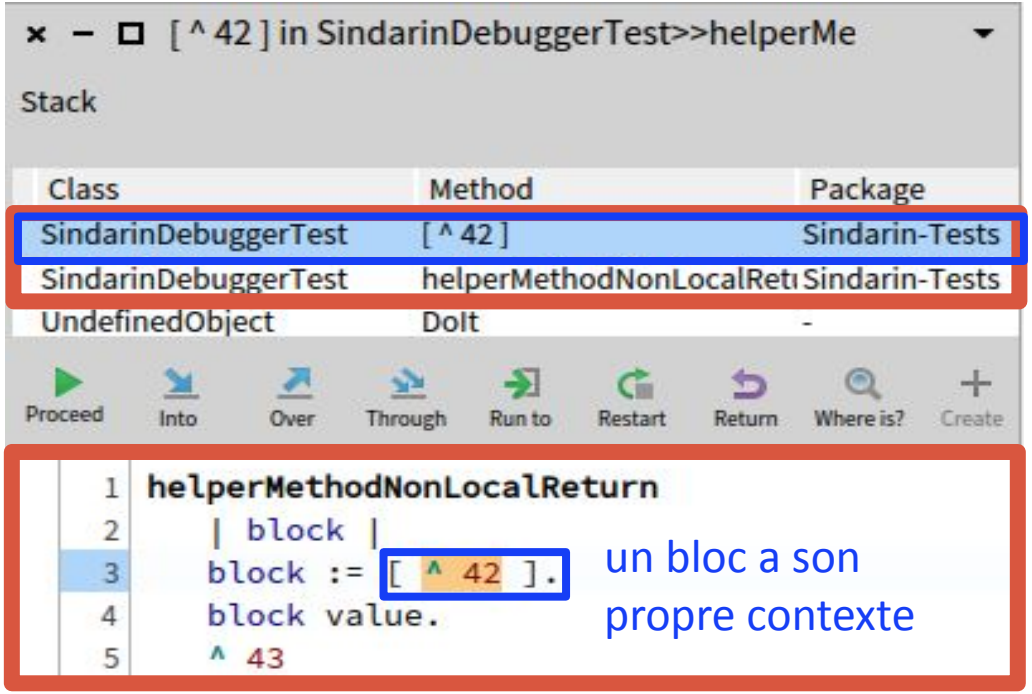
Proceed Into Over Through Run to Restart Return Whe

```
1 DoIt
2
3     ^ obj helperMethodNonLocalReturn
```

Sindarin : la commande stepToReturn, problème

ne rentre pas dans les sous-contextes

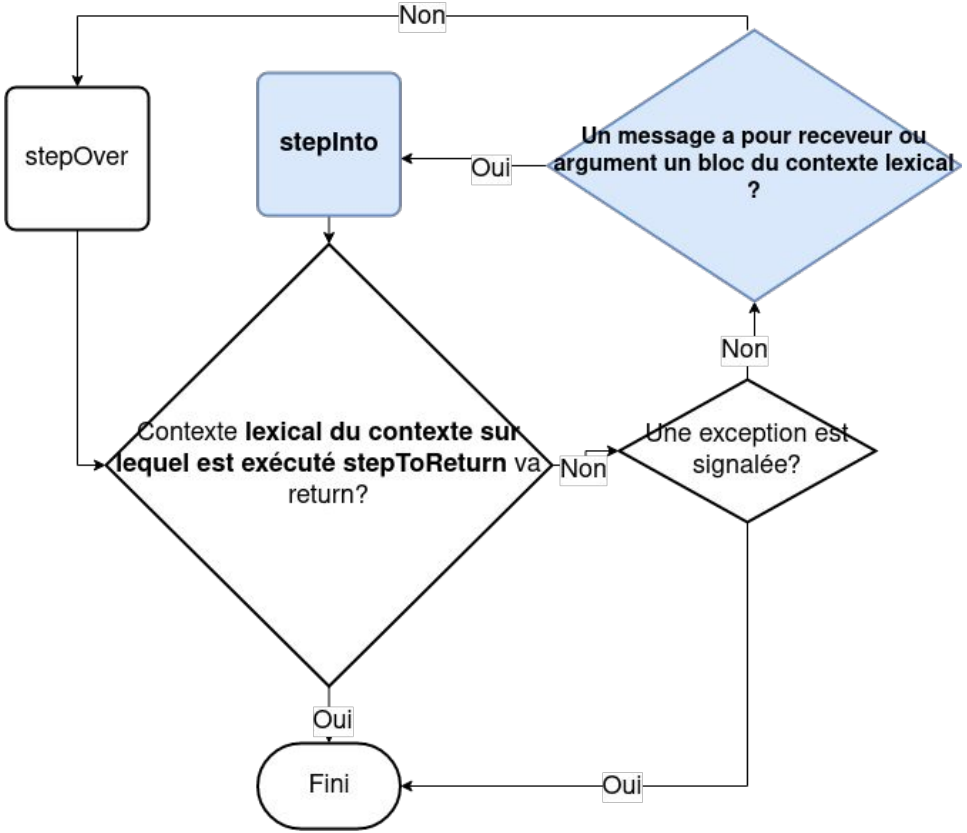




un bloc a son propre contexte

contexte lexical = tout ce qu'on voit à l'écran

Sindarin : la commande stepToReturn, solution



Initiation à la recherche

assister à des conférences à

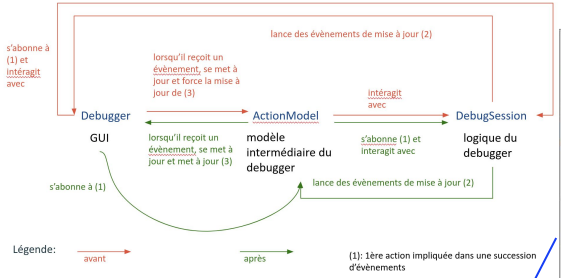
participation à des workshops

participation à

participation à la rédaction d’un papier de recherche

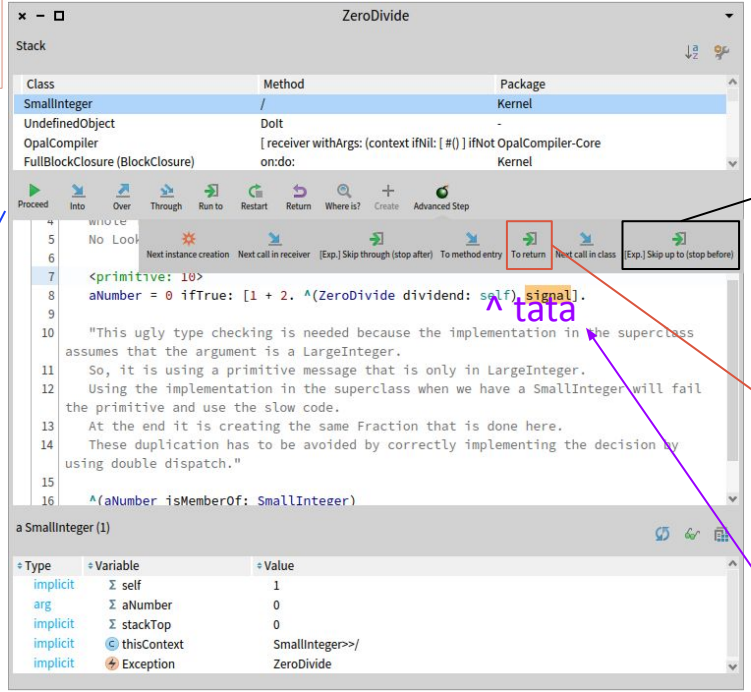
RECAP

11 PR : 8 intégrées, 3 en review



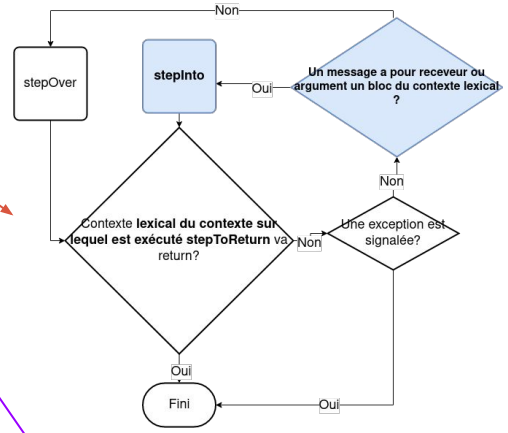
GUI | API

Séparation des préoccupations



^ tata

spécifique au bytecode



Towards Object-centric Time-traveling Debuggers

Maximilian Ignacio Willembrinck Santander[1], Steven Costiou[1], Adrien Vanègue[1] and Anne Etien[2]

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

Abstract
Object-centric debugging aims at facilitating the debugging of object-oriented programs by focusing debugging operations on specific objects. This technique is tedious to use because developers have to manually find objects to debug, which is not straightforward.

Time-traveling debuggers allows developers to explore executions back and forth in time. It has been shown that time-traveling features effectively facilitate debugging and program understanding.

We propose to combine these techniques to benefit from both of them to debug object-oriented programs. Time-travel navigation could help finding and remembering objects by providing means to explore executions back and forth. Object-centric debugging could extend time-traveling debugging with object-centric exploration features. These techniques have never been combined, and the challenges and benefits of such combination have never been explored.

We present *SeekerOC*, a time-traveling debugger prototype which provides object-centric debugging support. To combine both techniques, we use *Time-Traveling Queries*, a query system to automatically explore executions. We discuss the expected benefits of this combination, and we argue that exploring object-centric time-traveling debugging will open new research perspectives towards more effective debugging techniques and tools for object-oriented systems.

Keywords
Object-Centric Debugging, Time-Travel Debugging, Program Comprehension.

Rédaction d'un papier de recherche

tata := 1.

Bilan

- Découverte du debugging
- Découverte du (réel) fonctionnement de git, sur un projet open-source
- Découverte du domaine de la recherche
- Découverte de concepts complexes comme les bytecodes
- Poursuite en apprentissage : refonte de l’API Sindarin, remote debugging, object-centric debugging…

Chest, avant/après

Chests

Chests

ID	Name
1	Default

Chest Contents

Index	Object
1	42
2	43
3	toto

Do it Publish Bindings Versions Pages

```
Chest add: 42;
      add: 43;
      add: 'toto'
```

Remove Remove All Remove Remove All

Line: 1:1

an UndefinedObject (nil)

Raw Breakpoints Meta

Variable	Value
self	nil

```
self
```

Use Chests to store and access objects from anywhere

Chests

Chests Remove Remove All add

Default
titi

Chest Contents Remove Remove All

Name	Object
Default_3	42
Default_4	43
toto	toto

Chest Contents
Rename
Load Ctrl+L
Load all Shift+Ctrl+L
Inspect Ctrl+I

Do it Publish Bindings Versions Pages

```
Chest add: 42;
      add: 43;
      at: 'toto' put: 'toto'
```

Line: 1:1

a ByteString (toto)

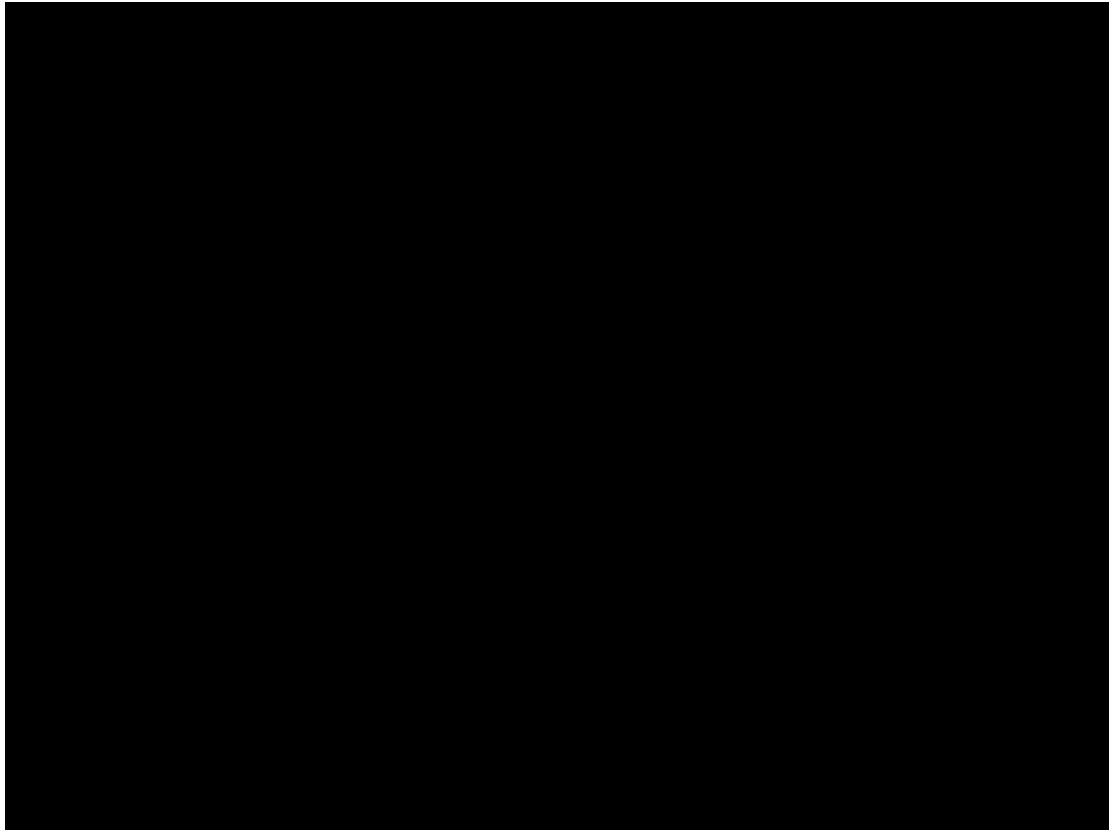
Preview Items Full Content Raw Breakpoints Meta

toto

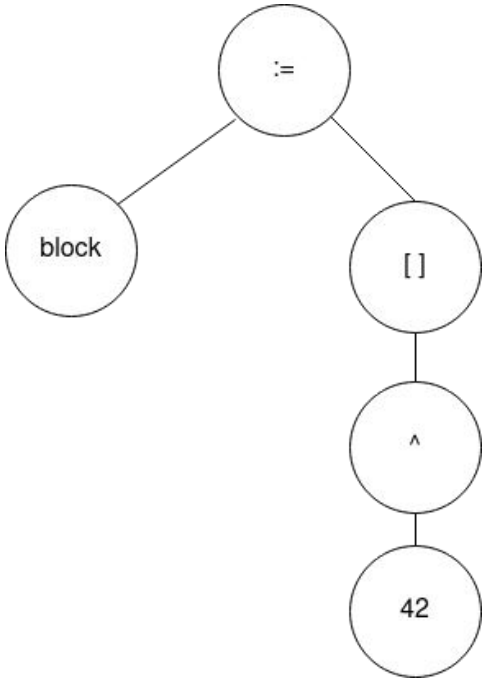
```
self
```

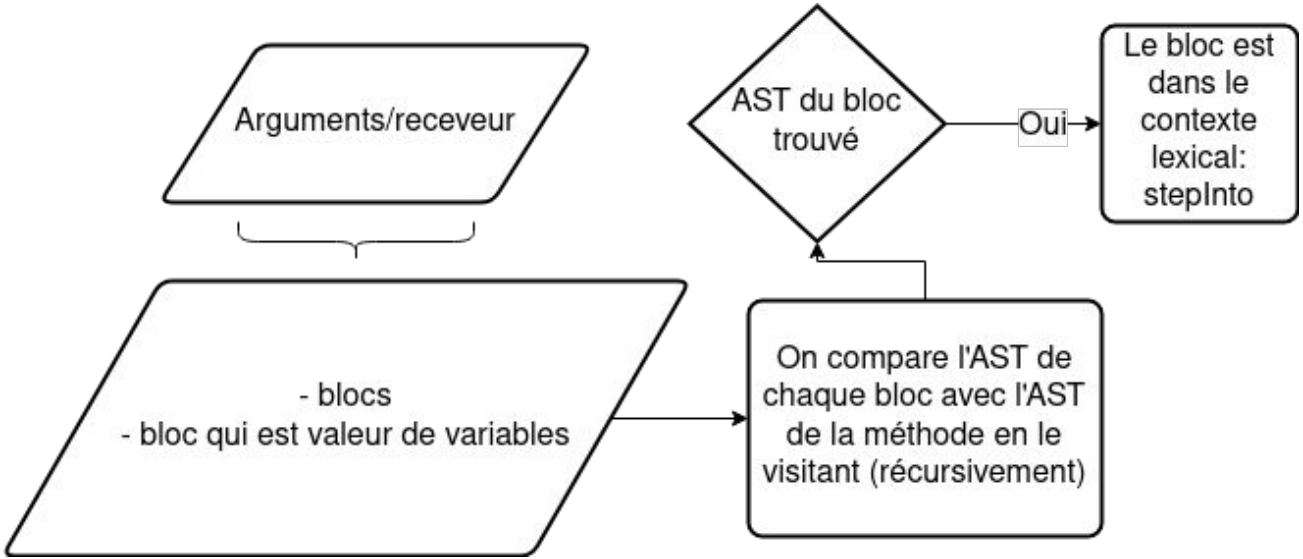
Use Chests to store and access objects from anywhere

Chest, demo



Sindarin, la commande stepToReturn : le bloc fait-il partie du contexte lexical?





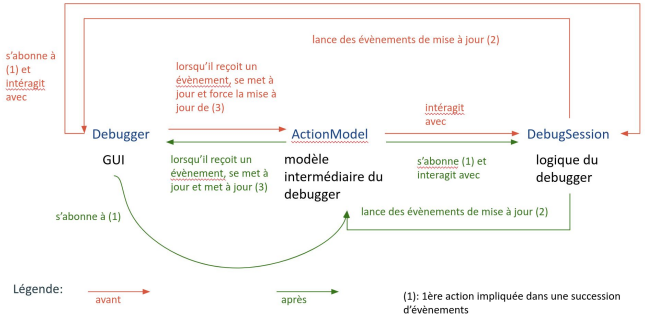
Conclusion

- 11 PR dont 8 intégrées dans Pharo à ce jour
- Responsabilité Unique + automatisation de l'action model = debugger plus extensible et maintenable
- Layout dynamique
- Sindarin: commande skip plus stable, convoitée par Thalès
- Sindarin: commande stepToReturn gère les cas les plus complexes utilisant les blocs
- Chest: refonte de l'IHM, plus intuitive + ajout de fonctionnalités déjà demandées
- Participation à la rédaction d'un papier de recherche

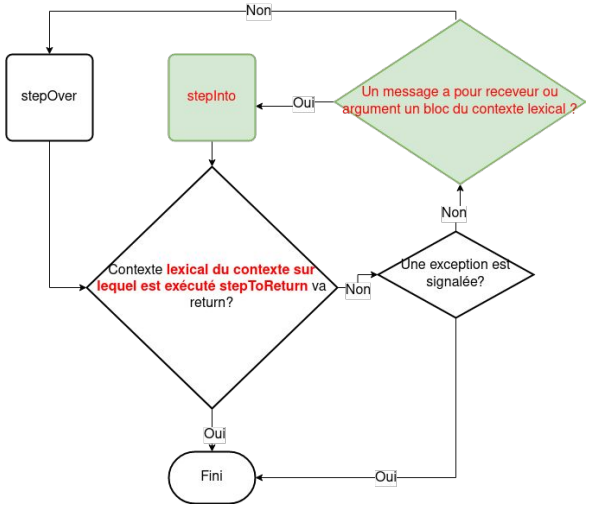
- Découverte du debugging
- 1ère expérience de contribution à un projet open-source
- Découverte du milieu de la recherche
- Approfondissements possibles:
  - Refonte de l'API Sindarin
  - Remote debugging
  - Object-centric debugging

RECAP

- Délégation des accès à la session vers l'action model

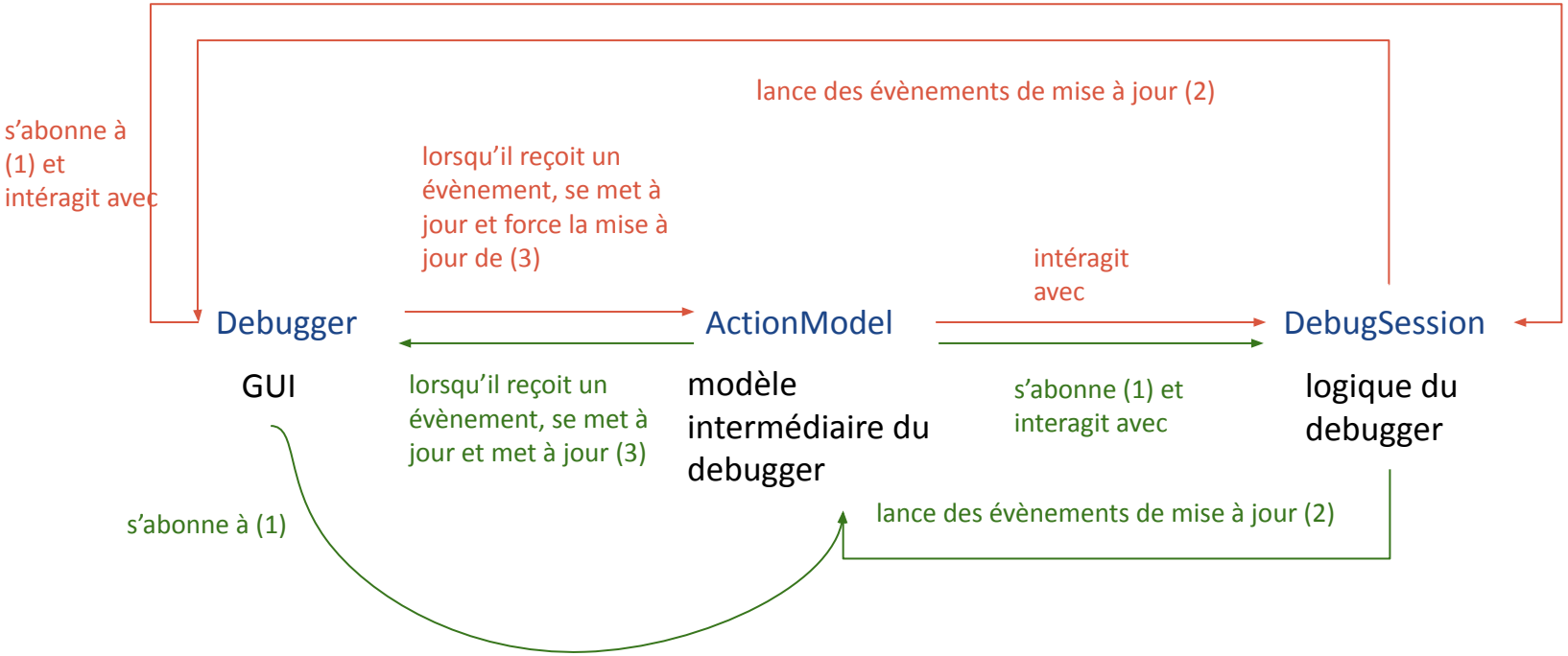


- Traitement spécifique de skip selon le bytecode impliqué



- Chest: refonte de l'IHM et de ses fonctionnalités
- Participation à la rédaction d'un papier de recherche

Séparation des préoccupations : refonte du système d'évènements



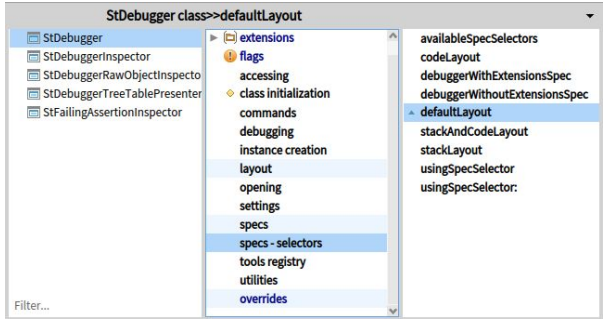
Légende:

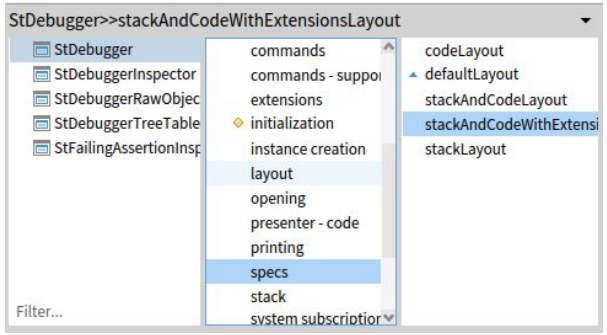
avant

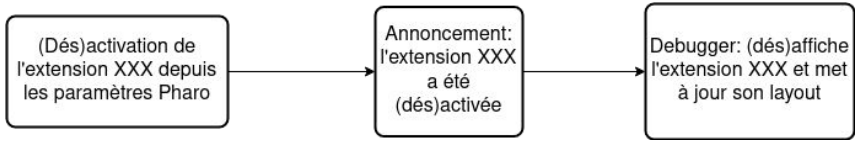
après

(1): 1ère action impliquée dans une succession d' évènements

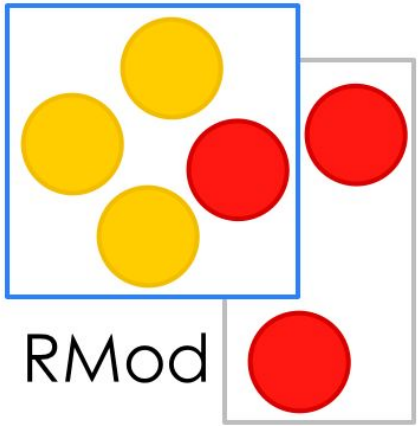
Affichage dynamique des extensions du debugger







Présentation de l’équipe RMOD



Séparation des préoccupations

accède à la session

Debugger

GUI

- implémente l'IHM du debugger
- délègue à l'action model les accès à la session

ActionModel

modèle intermédiaire du debugger

- implémente l'API du debugger
- se met à jour et met à jour le debugger après chaque action

DebugSession

logique du debugger

- opérations de stepping
- pile de contextes

Légende:

après