

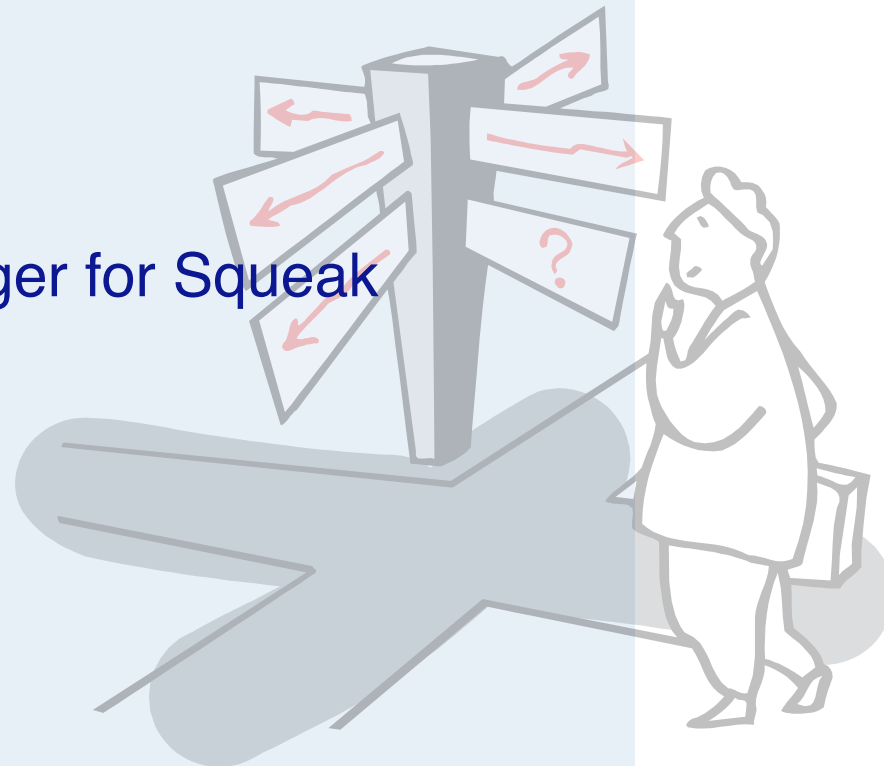
Design and Implementation of a Backward-In-Time Debugger

Christoph Hofer
Marcus Denker
Stephane Ducasse



Roadmap

- > Problem
- > Unstuck: A new Debugger for Squeak
- > Implementation
- > Lessons Learned
- > Future work



Problem

- > Debugger: Snapshot of state at time of error
- > Cause for errors is in the past
 - Who assigned *that* value?
- > very incomplete history available
 - guess were to set breakpoint, rerun

Example

```
Foo>>initialize
  var1 := 0.
  var2 := ''.
```

```
Foo>>start
  self beforeBar.
  self bar.
  self moreBar.
```

```
Foo>>beforeBar
  var1 = 0 ifTrue: [
    var2 := nil.]
```

```
Foo>>bar
```

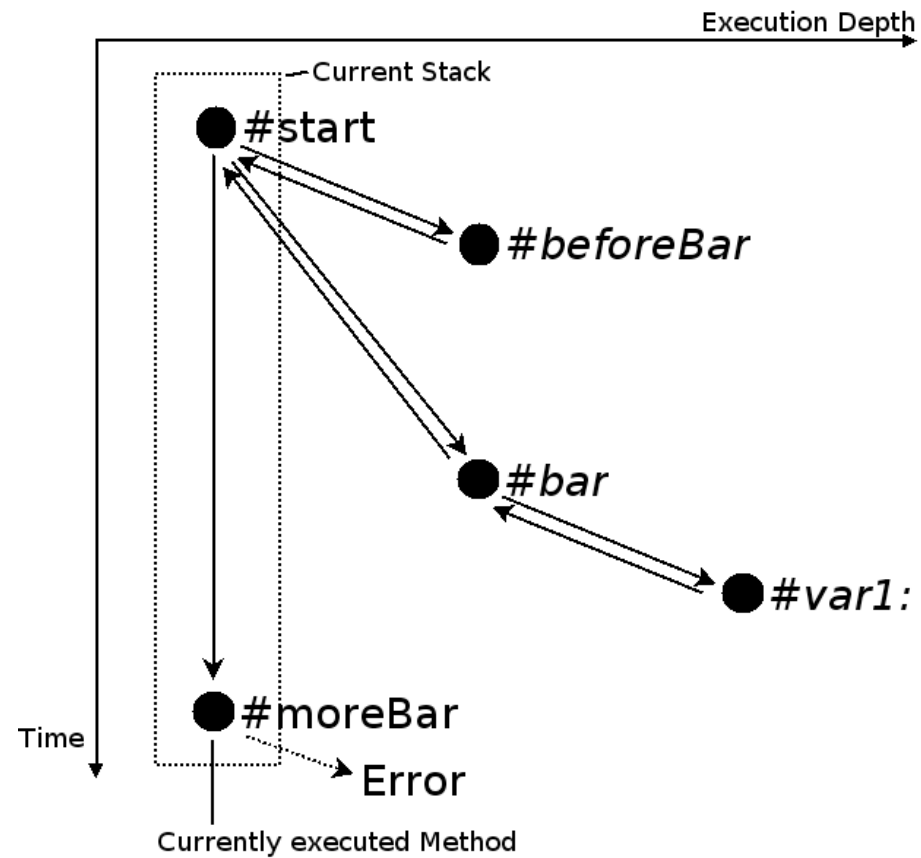
```
.....
```

```
Foo>>moreBar
```

```
  var2 size > 0 ifTrue:[
    ^var2 at: 1].
  ^''
```


Stack Trace

- > Squeak Debugger
- > Shows stack trace
 - methods not returned
 - old state lost



Methods in italic means their execution is finished

Solution



b
UNIVERSITÄT
BERN

- > We want:
 - Record the history of the program
 - View the state at any point in the past

- > Unstuck
 - A new Debugger for Squeak
 - Provides full trace information

Unstuck UI

1. Trace
2. Object
3. Code
4. History
5. Query

The screenshot shows the TraceDebugger application with the following components:

- Trace View (Left):** A list of events with a search bar at the top. The current event is highlighted in blue. The trace shows the execution of `JRecompiler#recompile`, which involves creating `RBMethodNode` objects and building a selector. A red '1' is placed next to the `RBMethodNode#selectorParts` event.
- Object View (Top Right):** Shows the state of the current object, including `selector` and `class`. A red '2' is placed next to the `selector` field.
- Code View (Middle Right):** Displays the source code of the `recompile` method. A red '3' is placed next to the `selector` parameter in the code.
- History View (Bottom Right):** Shows the sequence of events, including `event isSend &` and `event receiver class = CompiledMethod`. A red '5' is placed next to the `event receiver class` field.
- Search Bar (Bottom):** Contains the text "Search result: 11 events found".
- Navigation Controls:** Buttons for `first`, `next`, `previous`, `last`, `step forward`, `step backward`, `step into`, `step over`, `step out`, and `eofm` are visible.

Searching



u^b
UNIVERSITÄT
BERN

Variable	Search Domain
event	All events
send	all message sends
return	all method returns
varAccess	all variable accesses
instVarAccess	instance variable access
tempVarAccess	temporary variables

Searching: Example

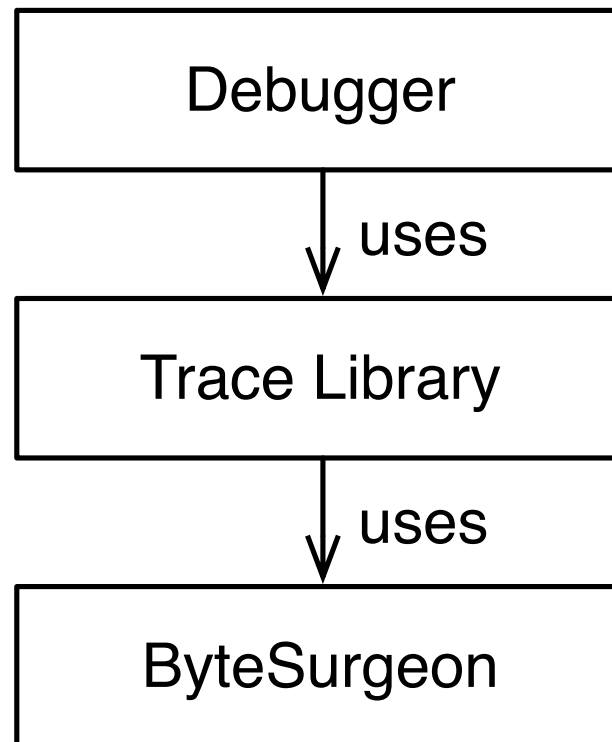
Query	Result
send selector = #foo	All the executed methods named "foo"
return returnValue > 4	All returns with a return value greater than 4
events isSend & (even arguments size = 1)	Message sends with exactly one argument

Coloring

- > We can assign a color to any object
- > Easy tracking of objects
- > Color is shown in all views of the UI

```
RBMethodNode class#basicNew -> a RBMethodNode
- a RBMethodNode#selectorParts:arguments:(an Array)
- a RBMethodNode#arguments:(#())
  #()#do:([] in RBMethodNode>>arguments: )
  a RBMethodNode#comments:(an OrderedCollection)
- a RBMethodNode#methodPatternStop -> a RBMethodNode
  a RBMethodNode#arguments -> #()
  #()#isEmpty -> true
  a RBMethodNode#selectorParts -> an Array({exampleInstVar(1,14,#(22))}
  an Array({exampleInstVar(1,14,#(22))})#first ->
  {exampleInstVar(1,14,#(22))}#stop -> 14
- a RBMethodNode#selector -> a RBMethodNode
  nil#isNil -> true
- a RBMethodNode#buildSelector -> #exampleInstVar(1,14,#(22))
  String class#new:(50)
```

Implementation



ByteSurgeon



b
UNIVERSITÄT
BERN

- > Framework for editing bytecode for Squeak
 - Like Javasist in Java, but:
- > Uses structural reflection to transform at runtime
 - Simple model: Inline code before / after a bytecode
 - Inlined code is normal smalltalk code
 - Not much knowledge about bytecode needed

Trace Library



b
UNIVERSITÄT
BERN

- > Called from annotated code
- > Builds up the trace

- > Provides
 - Trace model
 - Event pre-processing (ordering)
 - State reconstruction

State reconstruction

- > State not recorded for completely annotated classes
 - past state can be reconstructed from trace
- > System never completely annotated
 - Tracer saves state of non-annotated objects

Debugging system classes

- > Annotate classes used by Bytesurgeon or Tracer
 - System classes (e.g. Collection or String)
 - Compiler (e.g. AST)

- > Problems:
 - Classes used for annotation --> crash
 - Tracer records events generated by the tracer

Solution

- > Retain both methods (original + annotated)
- > Generate preamble
 - test for global
 - call original methods when inactive
- > Common problem when using reflection!
 - General solution?
 - Future work!

Benchmarks



UNIVERSITÄT
BERN

	Events	Slowdown	Memory (Kb)
Example	74	6	16
AST Bug	2725	3.8	800
Pier Trace	389689	248	88800

Future Work

- > further analyze + improve
 - Memory Consumption (GC effects)
 - Performance

- > Use behavioral reflection
 - fine grained selection
 - Scoping
 - Annotation of system classes

Conclusion

- > Problem of current debugging tools

- > Overview of Unstuck
 - UI
 - Implementation

- > Having the history available helps
 - Possible for small programs
 - Work needed for bigger systems + continuous use

Conclusion

u^b

b
UNIVERSITÄT
BERN

- > Problem of current debugging tools

- > Overview of Unstuck
 - UI
 - Implementation

- > Having the history available helps
 - Possible for small programs
 - Work needed for bigger systems + continuous use

Questions?