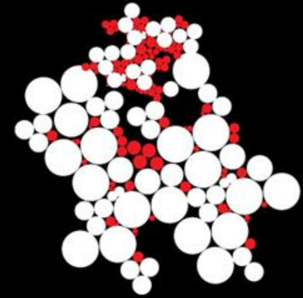
UNIVERSITY OF TWENTE.



A Fine-Grained Debugger for Aspect-Oriented Programming

Haihan Yin, Christoph Bockisch, Mehmet Akşit

{h.yin, c.m.bockisch, m.aksit}@cs.utwente.nl

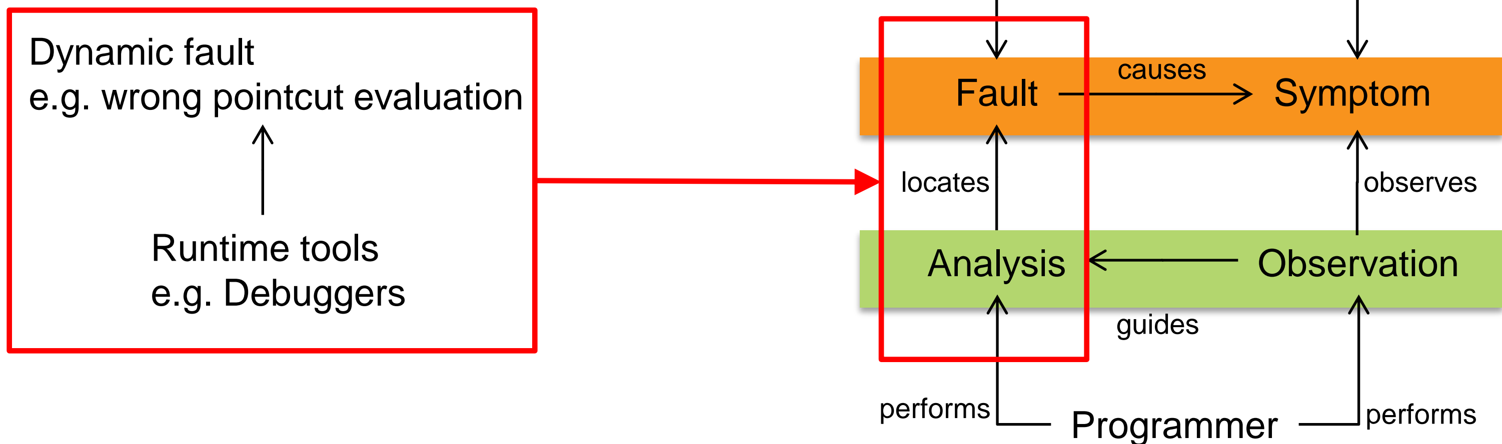
TRESE Software Engineering group
University of Twente, The Netherlands
http://trese.cs.utwente.nl

Why Debug AO Programs?

- Aspects can implicitly alter structure and behavior of other modules.
- Case study : 42 out of 102 AOP-related bugs are because of implicitness.

  - F. Ferrari et.al. An exploratory study of fault-proneness in evolving aspect-oriented programs. ICSE'10

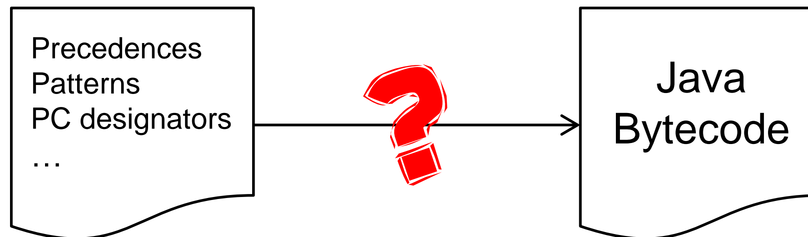
- Debuggers are used for analyzing and locating faults.



Haihan Yin {h.yin@cs.utwente.nl}

Limitations of Existing Debuggers

- AO programs are compiled to intermediate-representation (IR) of the base language
- IR partially or even entirely loses AO-information. In AspectJ,



- Debuggers that use debugging information stored in IR

- Java Debugger (JDB)

- AODA – W. De Borger et al. *A generic and reflective debugging architecture to support runtime visibility and traceability of aspects*. AOSD'09

- Wicca – M. Eaddy et al. *Debugger aspect-enabled programs.* SC'07

- TOD – G. Pothier and E. Tanter. *Extending omniscient debugging to support aspect-oriented programming.* SAC'08

Fault Categories & Symptoms

- Pointcut related
  - Incorrect pointcut composition
  - Incorrect pattern
  - Incorrect designator
  - Incorrect context

An advice is unexpectedly executed or not executed

- Advice related
  - Incorrect composition control
  - Incorrect flow change
  - Violated requirements

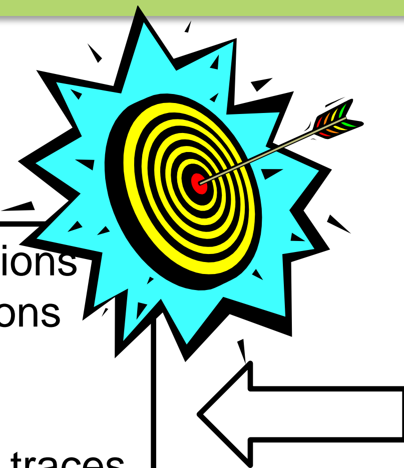
An advice does not behave as expected

Observations & Debugging Tasks

Setting AO breakpoint

- Evaluating pointcut sub-expressions
- Evaluating pattern sub-expressions
- Flattening pointcut references
- Inspecting runtime values
- Inspecting AO-conforming stack traces
- Inspecting program compositions
- Inspecting precedence dependencies
- Excluding and adding AO definitions

Locating AO constructs



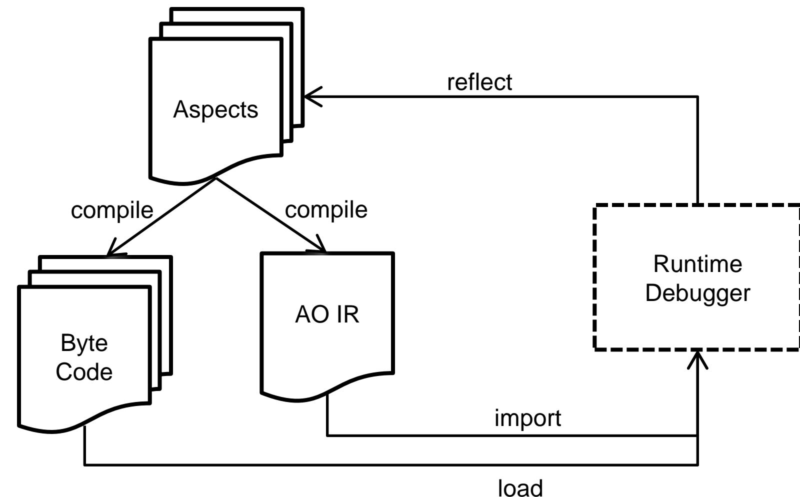
An advice is unexpectedly executed or not executed

An advice does not behave as expected

Aspect-Oriented Intermediate Representation

Source code :

```
aspect Azpect {
  before (Base b) :
    call(* Base.foo()) &&
    target(b)
  { … }
}
```

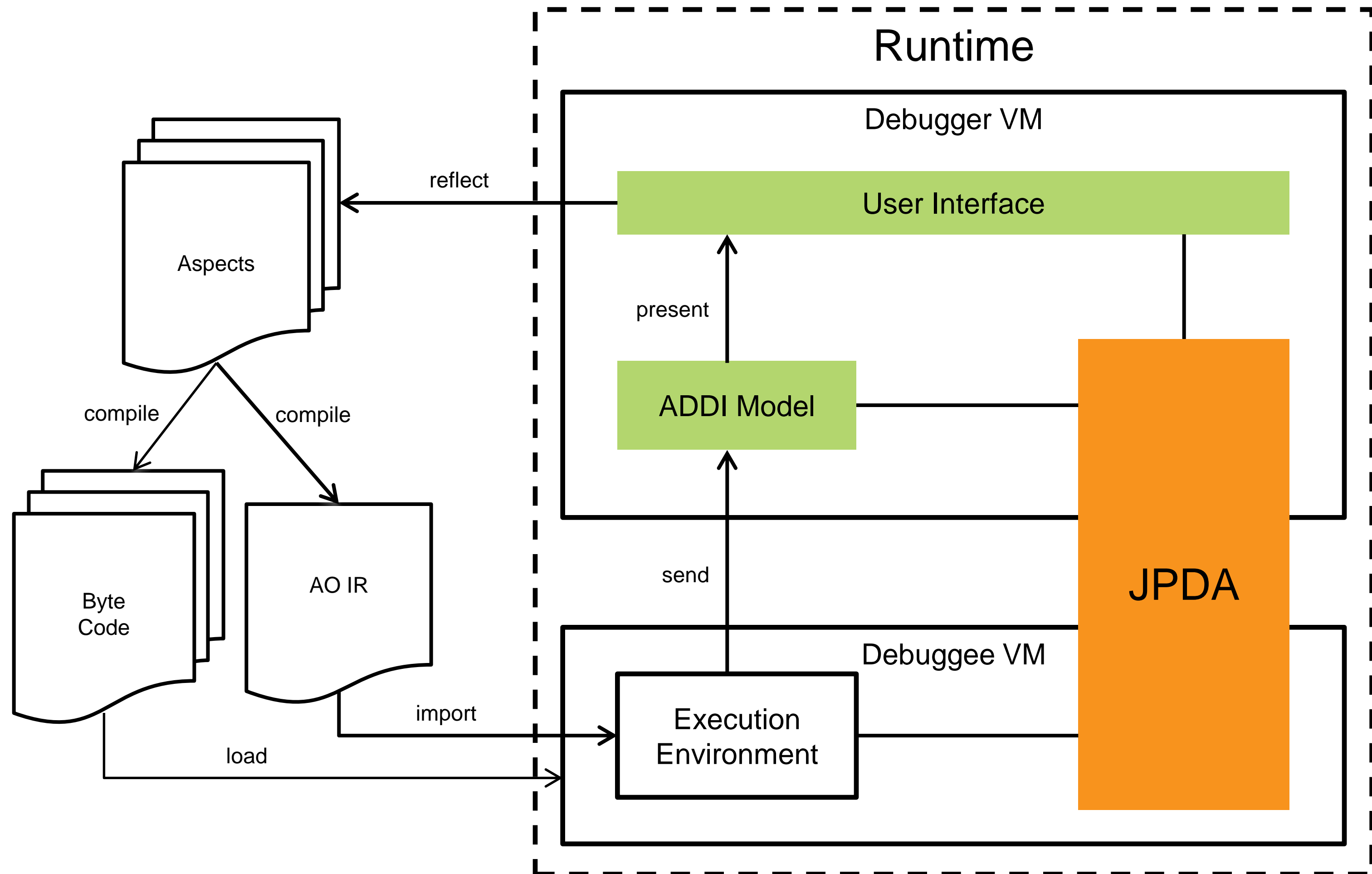


AO IR in form of XML:

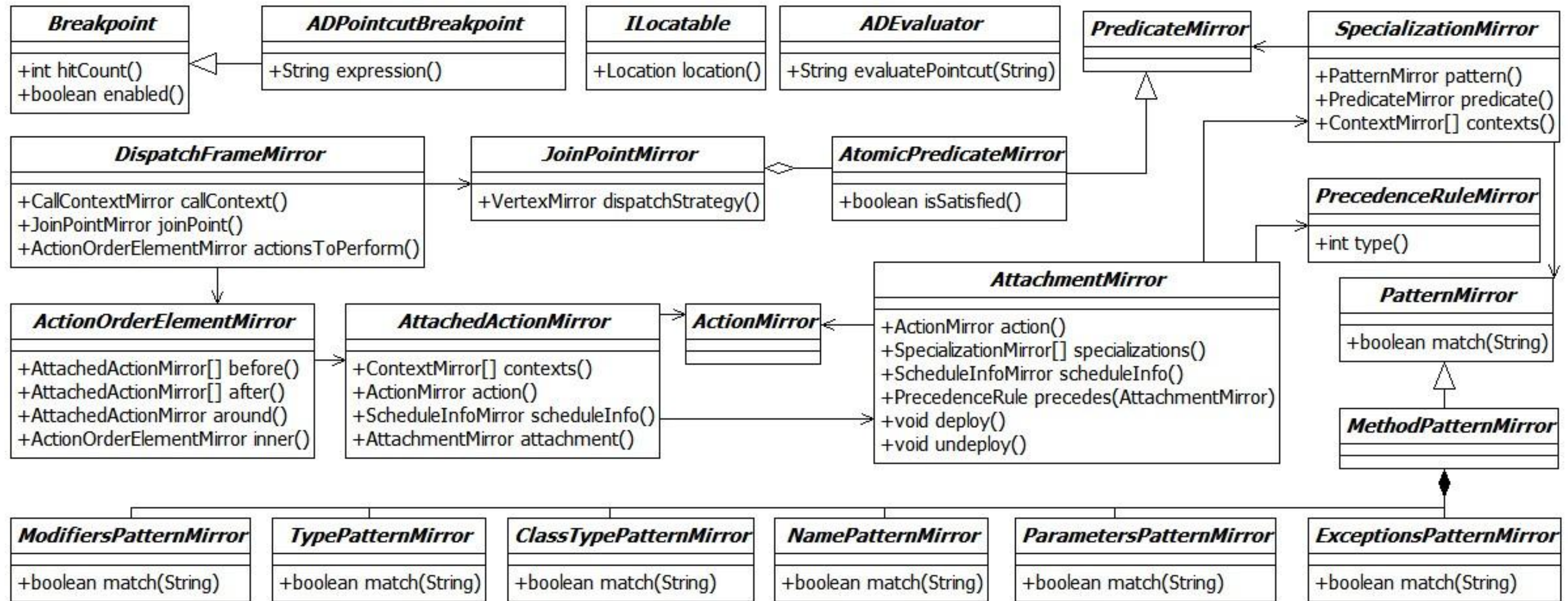
```
<attachment>
<specialization>
...
<atomicPredicate type="InstanceofPredicate">
   <requiredTypeName location=... >Base</requiredTypeName>
   <context type="CalleeContext" location=... />
</atomicPredicate>

<context type="CalleeContext" location=... />
...
</specialization>
</attachment>
```

The Debugger Architecture



Advanced-Dispatching Debug Interface



Evaluating pattern sub-expression

Inspecting precedence dependencies

Excluding and adding AO definitions

Inspecting program compositions

The Graphical Representation of Dispatch

```
1  public class Base {
2    public static void main(String [] args) {
3      Base b = new BaseSub();
4      b.advicedMethod();
5     }
6    public void advicedMethod() {
7      // …
8    }
9  }
10 public class BaseSub extends Base { // … }
```

AtomicPredicate
Callee is a (sub)type of test.BaseSub

True

Action
java.lang.Object aspects.Azpect.around@line9()
void test.Base.advicedMethod()

False

Action
void aspects.Azpect.before@line6()
java.lang.Object aspects.Azpect.around@line9()
void test.Base.advicedMethod()

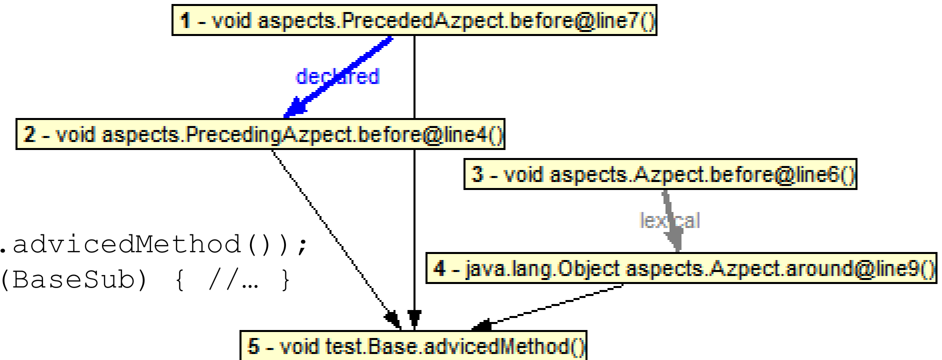
```
11   public aspect Azpect {
12   pointcut base() : call(* Base.advicedMethod());
13   before() : base() && !target(BaseSub) {
14     //…
15   }
16   Object around() : base() {
17      proceed();
18      return null;
19   }
20  }
```

Bug: Wrong negation operator

The Graphical Representation of Precedence Dependencies

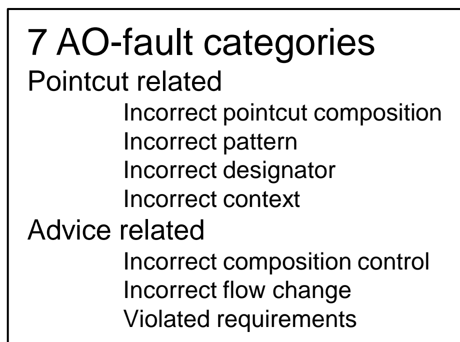
```
1   aspect PrecedingAzpect {
2     before() : call(* Base.advicedMethod()) { //... }
3   }
4   aspect PrecededAzpect {
5     before() : call(* Base.advicedMethod()) { //... }
6   }
7   aspect PrecedenceAzpect {
8     declare precedence : PrecededAzpect, PrecedingAzpect;
9   }
```

Bug: Wrong precedence

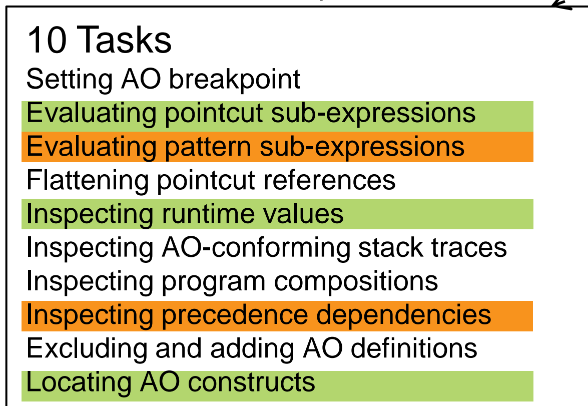


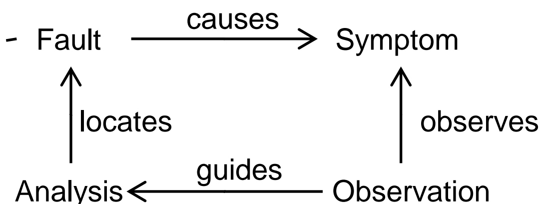

```
10  public aspect Azpect {
11  pointcut base() : call(* Base.advicedMethod());
12    before() : base() && target(BaseSub) { //… }
13    Object around() : base() {
14      proceed();
15      return null; }
16  }
```

Summary



locates





realizes