Plan for change! Or how a lack of modularity hinders Soot to reach its true potential

Eric Bodden

PADERBORN UNIVERSITY
The University for the Information Society

Fraunhofer
IEM

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE
SOFTWARE ENGINEERING
GROUP

What is Soot?

- a free compiler infrastructure, written in Java (LGPL)
- was originally designed to analyze and transform Java bytecode
- original motivation was to provide a common infrastructure with which researchers could compare analyses (points-to analyses)
- has been extended to include decompilation, visualization, Android support, inter-procedural analysis support, etc. etc.

What is Soot now?

Current main applications:

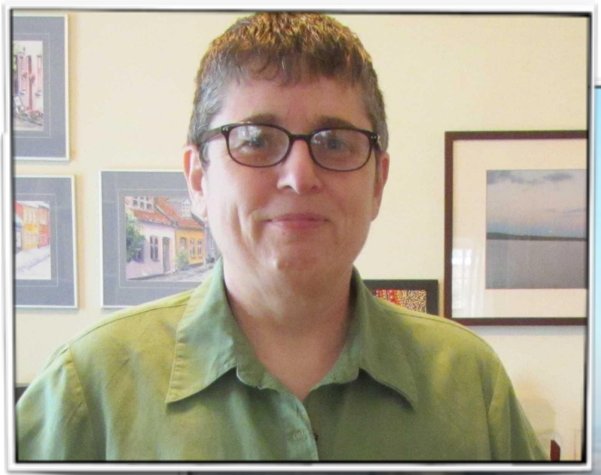
- Basis for prototyping new static-analysis and dynamic-analysis algorithms
- Basis for special-purpose analysis tools
- Currently most analyses probably rather for Android than Java

Soot Past and Present

- Started in 1996-97 with the development of coffi by Clark Verbrugge and some first prototypes of Jimple IR by Clark and Raja Vallée-Rai
- First publicly-available versions of Soot 1.x were associated with Raja's M.Sc. thesis
- New contributions and releases have been added by many researchers from around the world
- Currently maintained by my research group at Darmstadt and Paderborn

Soot & me (2003)

Soot & me (2006)



Thinks to like about Soot

- The Jimple IR
  - Typed, stackless 3-address code
- Analyses based on Jimple
  - Mainly: Call-graph construction, points-to analysis
  - Many clients: typestate, race detection, slicing, taint analysis, performance analyses, etc. etc.

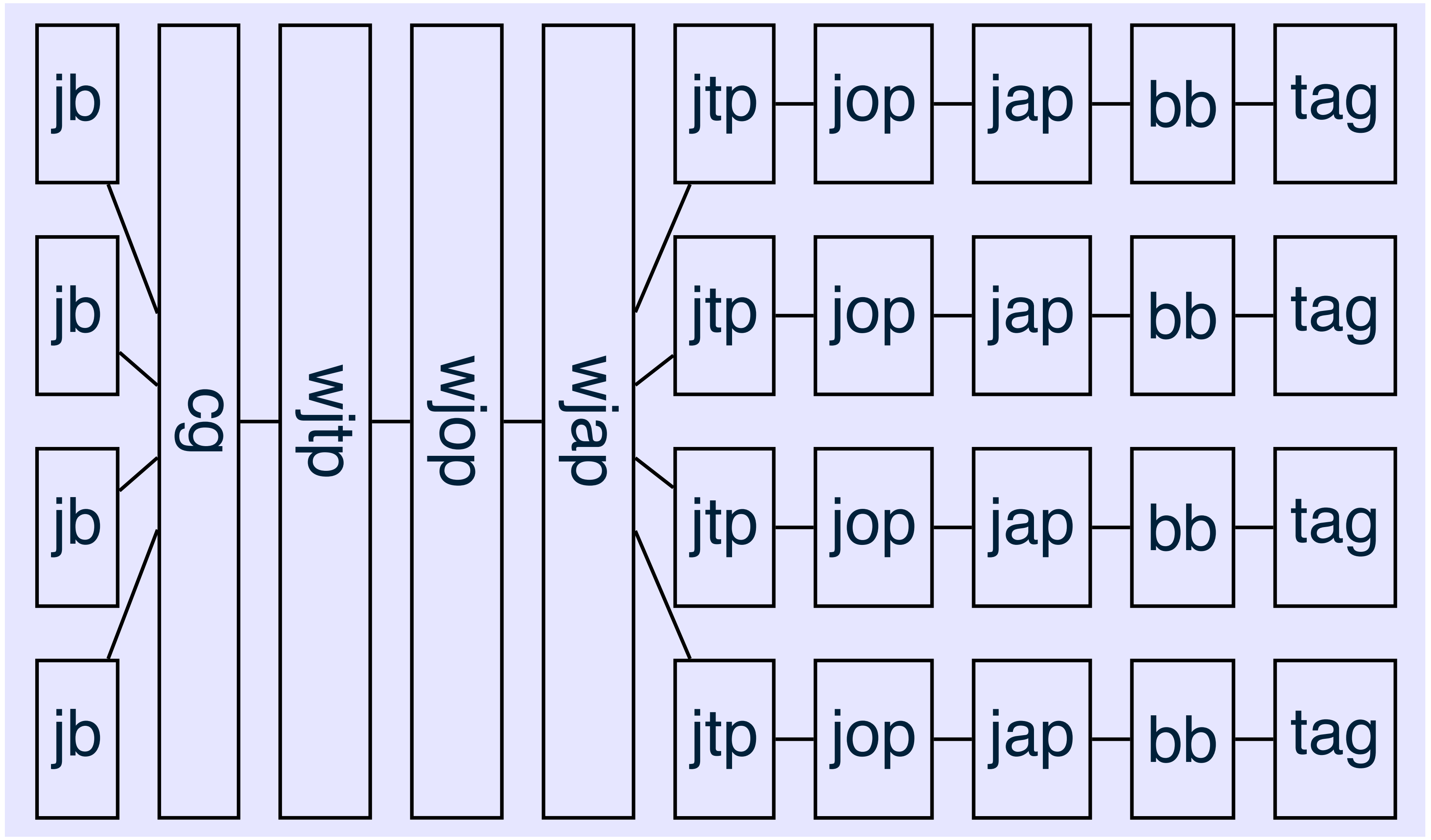
Thinks to like about Soot

Also: everything's so easy to access!

```
Scene.v().getSootClass(name)

Scene.v().getMainClass()

Scene.v().getEntryPoints()

Scene.v().getActiveHierarchy()

...
```

Thinks to like about Soot

Soot's always in control



Thinks to like about Soot

Instrumentation really is a piece of cake:

```
Chain stmts = methodBody.getUnits();
stmts.insertBefore(oldStmt, newStmt);
```

Things one learns to dislike about Soot

- The Jimple IR
  - Because everything depends on it
  - Because it's construction is slow
- My wish: An extensible, fast to compute IR with explicitly declared assumptions and dependencies

Things one learns to dislike about Soot

Everything's so easy to access!

```
Scene.v().getSootClass(name)

Scene.v().getMainClass()

Scene.v().getEntryPoints()

…
```

Things one learns to dislike about Soot

Everything's so easy to access!

- Problem: Everything depends on the scene; strong coupling throughout
- Soot 2.0 introduced way to *reset all singletons*

Things one learns to dislike about Soot

What I would like:

- Modularly composable analyses
  - Through Dependency injection (?)
- No global state, explicit passing of all state
- More easily supports incremental updates etc.

Things one learns to dislike about Soot

Soot's always in control

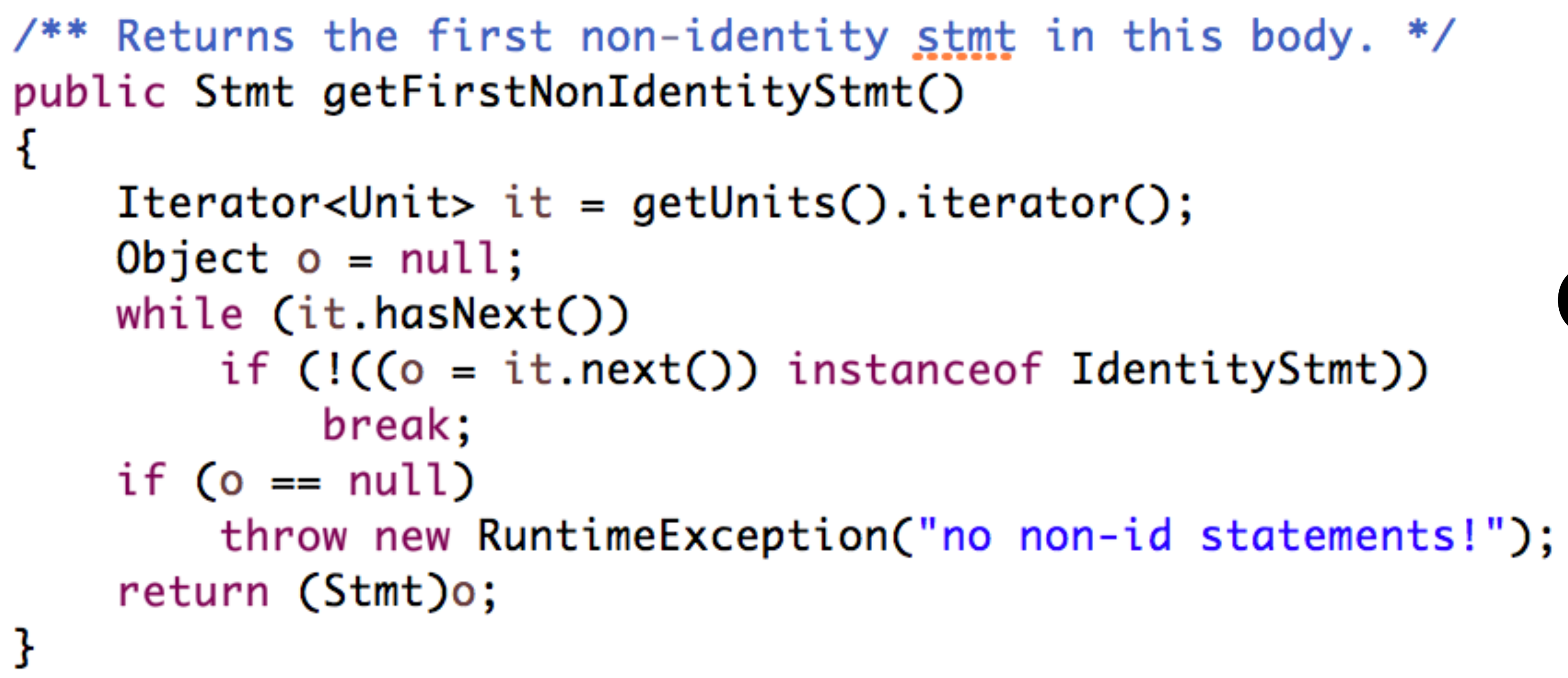
What if e.g. an IDE should be in control?

Hence maybe I'd actually prefer if Soot were a library instead of a framework.

→ No inversion of control

Things one learns to dislike about Soot

Ability to instrument makes things slow:



```
/** Returns the first non-identity stmt in this body. */
public Stmt getFirstNonIdentityStmt()
{
    Iterator<Unit> it = getUnits().iterator();
    Object o = null;
    while (it.hasNext())
        if (!((o = it.next()) instanceof IdentityStmt))
            break;
    if (o == null)
        throw new RuntimeException("no non-id statements!");
    return (Stmt)o;
}
```

O(|Stmt|)

Things one learns to dislike about Soot

Ability to instrument makes things slow:

Profiling revealed that lots of time is spent in such operations, which are useless for folks who only want to do static analysis.

I want the common case to be fast, uncommon case to be possible.

My wish list for a "Soot 3.0"

Extensible, flexible IR, created "on-demand"

no global state
explicit passing of state

no inversion of control
i.e. client is in control

no performance compromises due to instrumentation

Prof. Dr. Eric Bodden
Chair for Software Engineering
Heinz Nixdorf Institut
Zukunftsmeile 1
33102 Paderborn

Telefon: +49 5251 60-3313
eric.bodden@uni-paderborn.de

https://www.hni.uni-paderborn.de/swt/

https://blogs.uni-paderborn.de/sse/