The Soot framework for Java program analysis: a retrospective

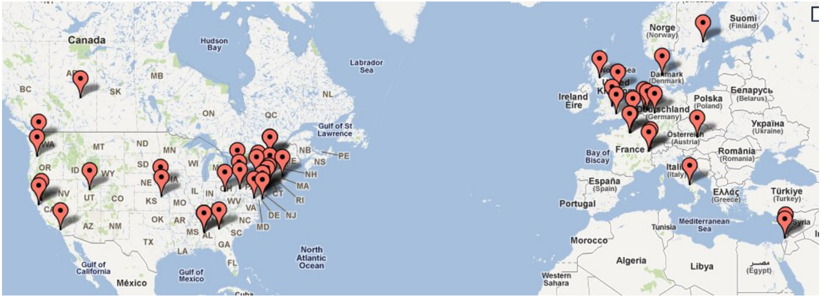
Patrick Lam, Eric Bodden, Ondřej Lhoták, and Laurie Hendren

October 2011

Soot

a compiler framework for Java (bytecode), enabling development of static analysis tools.

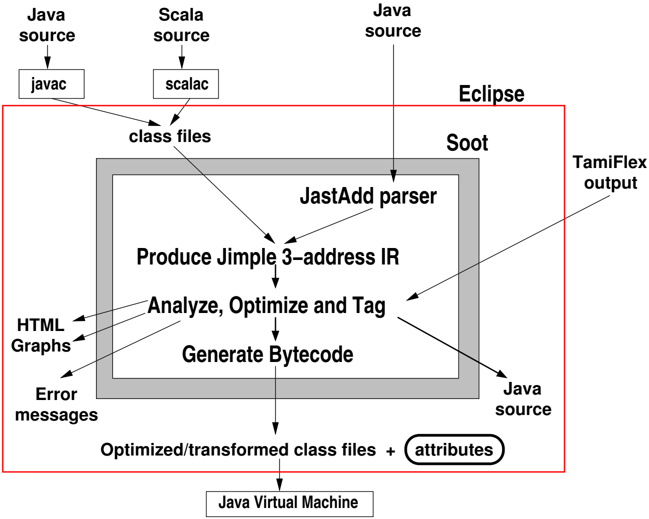
Map of Reported Soot Users



Outline

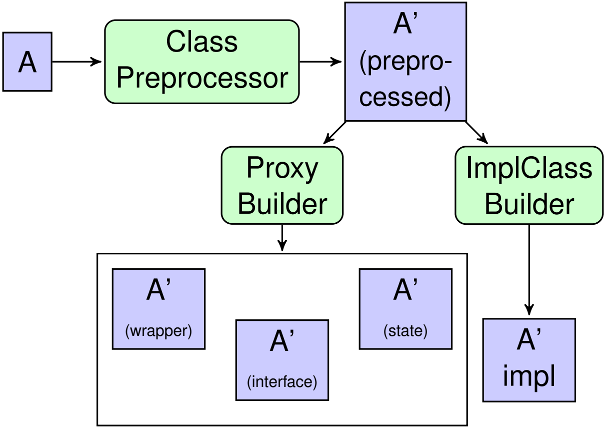
- About Soot
- About Soot's development

Soot Workflow



Case Study: A Soot Application

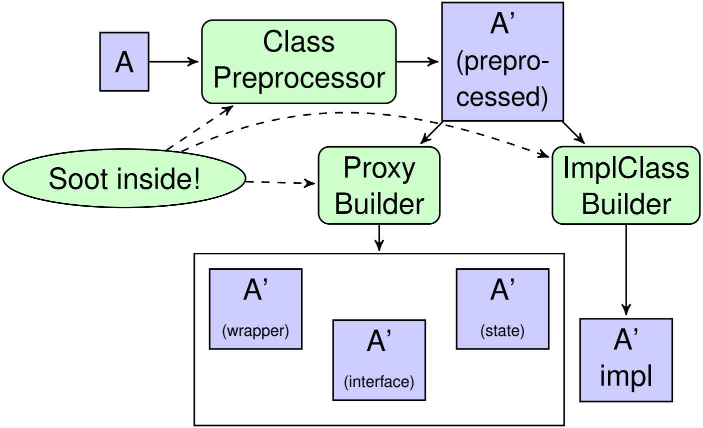
A. Orso and A. Rao and M. J. Harrold. “A Technique for Dynamic Updating of Java Software”. ICSM 2002.



The preprocessor e.g. converts field reads into method calls.

Case Study: A Soot Application

A. Orso and A. Rao and M. J. Harrold. “A Technique for Dynamic Updating of Java Software”. ICSM 2002.



The preprocessor e.g. converts field reads into method calls.

More Selected Soot Applications

- Analysis of Concurrent Programs
- Symbolic Execution
- Combined Static and Dynamic Analysis Approaches (static part, plus instrumentation for dynamic analysis)

Part I

About Soot

We start by describing Soot's features, namely:

- intraprocedural features;
- interprocedural features; and
- getting results out of Soot.

Intraprocedural Features

- Provides three-address code.
- Supports implementing dataflow analyses.

Three-Address Code

```
public int foo(java.lang.String) {
  // [local defs]
  r0 := @this;                // IdentityStmt
  r1 := @parameter0;

  if r1 != null goto label0;  // IfStmt

  $i0 = r1.length();          // AssignStmt
  r1.toUpperCase();           // InvokeStmt
  return $i0;                 // ReturnStmt

label0:
  return 2;
}
```

Connecting with Java source

Each Jimple statement

```
if r1 != null goto label0; // IfStmt
```

belongs to:

- a SootMethod, e.g. `foo(String)`, and
- a SootClass, e.g. `Foo`,

reflecting the structure of the original source code.

You can also get:

- line number information (if available), e.g. “`Foo.java:72`”.
- original variable names (on a best-effort basis).

Dataflow Analysis Example: “Live Locals”

Question:

At a given program point *p*, which locals *v* will be accessed in the future?

```
void foo(boolean b) {
  int x = 5, y = 2;

  System.out.println(x);
  if (b) {
    x = bar(y*2);
  } else {
    foo(false);
  }
  System.out.println(x);
}
```

Dataflow Analysis Example: “Live Locals”

Question:

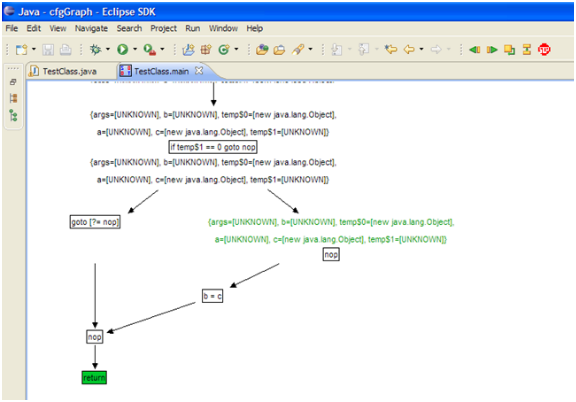
At a given program point *p*, which locals *v* will be accessed in the future?

```
void foo(boolean b) {
  int x = 5, y = 2;                          // {x, y, b}

  System.out.println(x);                     // {x, y, b}
  if (b) {                                      // {x, y}
    x = bar(y*2);                                 // {x}
  } else {
    foo(false);                                   // {x}
  }
  System.out.println(x);                           // {}
}
```

Dataflow Analysis Example: “Live Locals”

Soot’s Eclipse plugin helps you debug your flow analysis.



Interprocedural Features

- Call graph/pointer information
- (Side effect analysis)
- (Reflection)

Why Call Graphs?

Sophisticated static analyses need to answer questions like:

```
foo() {
    A o = ...;
    o.bar();
}
```

```
class A {
    bar() {
        /* */
    }
}
class B extends A {
    bar() {
        /* */
    }
}
```

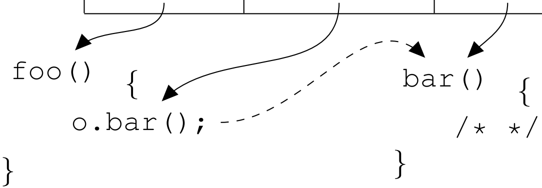
"Which methods might `o.bar()` reach?"

Call Graphs in Soot

Spark (part of Soot) computes call graph edges, which contain:

- Source method
- Source statement (if applicable)
- Target method
- Kind of edge

source m.	source stmt.	target m.	kind
•	•	•	VIRTUAL



Points-to Analysis

A closely related question:

Could `x` and `y` be aliases in:

```
x.f = 5;
y.f = 6;
z = x.f;
```

Spark can answer this question with a call to `hasNonEmptyIntersection()` on points-to sets.

Soot Output

There are many ways to get results out of Soot:

- *abc*: reads Java and AspectJ source, produces Java bytecode.
- *model checking*: generate summaries (in Java bytecode plus modelling primitives) of system environment behaviour.
- *tracematch/race condition detection*: generate error messages or warnings.
- *side-effect information*: generate attributes encoding the information along with the Java bytecode.

Running unaltered versions of Soot

Use Soot as a:

- disassembler to three-address code;
- bytecode optimizer; or
- visualizer for CFGs and analysis results, in Eclipse.

Extending Soot

You can write a compiler pass extending Soot, as either a

- `BodyTransformer`, for a intraprocedural analysis; or
- `SceneTransformer`, for a whole-program analysis.

You choose where this pass should run by putting it in a `Pack`.

Use `Map`s or attributes to share analysis results.

We explicitly disallow subclassing of IR statements, based on past experience. (Mixins would be OK).

To run extended Soot, you create a custom main class which calls `soot.Main.main()`.

Part II

About Soot's Development

History

Initial release in 1999–2000; Soot 1.0.0 was an intraprocedural Java bytecode analysis framework.

Soot Evolution

(credit: persocomholic/flickr)

Stepwise evolution of key features:

1. Local variable type inference, initially by Gagnon et al; later by Bellamy et al.
2. Call graph information, initially Variable Type Analysis by Sundaresan et al; subsumed by Spark.

Support and Community

(credit: Marsyas/Wikimedia Commons)

- Main agora: Soot mailing list, about 30 messages/month.
- Soot Bugzilla contains some bugs.
- Soot Wiki is good for recording certain types of information.
- Publicly readable Subversion repository; we'd welcome external committers.

Licensing

Soot is licensed under GNU Lesser General Public License.
We recommend choosing a license that works for you.

- McLab (compiler framework for MATLAB) will be released under the Apache 2.0 license.

Documentation

Documentation is critical to framework success.

- API carefully designed.
- Some Javadoc documentation.
- Soot tutorials.
- Soot Survivor's Guide by Einarsson and Nielsen.
- Plus: Helpful error messages.

Future Improvements for Soot

Some future directions where we'd like to see Soot improvements:

- faster startup and computation time;
- structured interprocedural analysis support;

Future Improvements for Soot

(credit: wwarby/flickr)

Some future directions where we'd like to see Soot improvements:

- faster startup and computation time;
- structured interprocedural analysis support;

Future Improvements for Soot

(credit: Mike Hunt/Wikimedia commons)

Some future directions where we'd like to see Soot improvements:

- faster startup and computation time;
- structured interprocedural analysis support;

Reflections

Soot does what we expected it to do.

- a surprise: unsound and incomplete analyses.

Challenges:

- keeping up with external changes (e.g. in the Java specification);
- incorporating external extensions into Soot.

Useful Features for Compiler Frameworks

While Soot doesn't have these features, they are indispensible for compiler frameworks.

- some way of avoiding redundant re-computations, e.g. incremental computation;
- quasiquoting, for easily generating code from templates.

Reflections on Compiler Frameworks

Our suggestions for compiler frameworks and the community:

- make it easy to independently release extensions (non-monolithic structure, like CPAN);
- the community must value software and data releases;
- we need more venues for framework papers.

Reasons for Success

Soot:

- provided the right features at the right time;
- was easy enough to use (availability, license, community).

Key features:

- Jimple intermediate representation;
- Spark pointer analysis toolkit.

Thanks!

Soot's development was supported in part by:

- Canada's Natural Science and Engineering Research Council
- Fonds de recherche du Québec—Nature et technologies
- IBM's Centre for Advanced Studies, and an Eclipse Innovation Grant.

Eric Bodden is supported by CASED (www.cased.de).

Contributors

Initial Designer:

Raja Vallée-Rai

Maintainers:

Patrick Lam, Feng Qian, Ondřej Lhoták, Eric Bodden

Project Advisor:

Laurie Hendren

Contributors:

Ben Bellamy	John Jorgensen	Chris Pickett
Will Benton	Felix Kwok	Patrice Pominville
Marc Berndl	Patrick Lam	Feng Qian
Eric Bodden	Jennifer Lhotak	Hossein Sadat-Mohtasham
Phong Co	Ondrej Lhotak	Ganesh Sittampalam
Archie Cobbs	Lin Li	Manu Sridharan
Torbjorn Ekman	Florian Loitsch	Vijay Sundaresan
David Eng	Jerome Miecznikowski	Julian Tibble
Etienne Gagnon	Antoine Mine	Navindra Umanee
Chris Goard	Nomair Naeem	Raja Valée-Rai
Richard Halpert	Matthias Perner	Clark Verbrugge

External contributors

- Ben Bellamy at Oxford (type assigner);
- Torbjörn Ekman at Oxford (Java 5 parser);
- Manu Sridharan, while at Berkeley (demand-driven pointer analysis).

Notable Changes in Soot

Over the years, we and others have improved Soot:

- a single singleton;
- dealing with partial programs;
- better front-end parsers;
- demand-driven efficiency improvements.

List of Soot Users

- McGill University, 3605, rue de la Montagne, Montreal, QC H3G 2M1, Canada
- Rutgers University, United States
- University of Washington, United States
- University of Alberta, Canada
- Georgia Tech, Atlanta, GA, USA
- Portland State University, Portland, OR 97201, USA
- Imperial College London, United Kingdom
- Rensselaer Polytechnic Institute, Troy, NY 12180, USA
- The Ohio State University Airport, United States
- Allegheny College, 520 N Main St, Meadville, PA 16335, USA
- University of Alabama, United States
- University of Warwick, CV8, UK
- Dortmund University of Technology, August-Schmidt-Straße 4, 44227 Dortmund, Germany
- Kansas State University, Manhattan, KS 66502, USA
- Drexel University, Philadelphia, PA 19104, USA
- Brigham Young University, 350 Clyde Bldg N, Provo, UT 84602, USA
- University of Buenos Aires - Buenos Aires, Capital Federal, Argentina
- University of Waterloo, Canada
- UC Berkeley, Oakland, CA, USA
- University of Maryland
- Hawthorne, NY, USA
- University of Aarhus, Birk Centerpark 15, 7400 Herning, Denmark
- imec Ieper, Ter Waarde 44, 8900 Ypres, Belgium
- MIT, Cambridge, MA, USA

List of Soot Users II

- University of Pittsburgh, Pittsburgh, PA, USA
- Strathclyde University, University of Strathclyde, Glasgow, Glasgow City G4 0, UK
- Uppsala, Sweden
- University of California Davis, United States
- Rocquencourt, France
- Cornell University, Ithaca, NY, USA
- Paris, France
- University of Delaware, Lewes, DE 19958, USA
- Radboud University, Comeniuslaan 4, 6525 HP Nijmegen, The Netherlands
- University of Geneva, Rue du Général- Dufour 24, 1211 Genève 4, Switzerland
- University Medical Center Utrecht, 3584 CX Utrecht, The Netherlands
- Victoria University of Wellington, Rutherford House Level 5/23 Lambton Quay, Pipitea 6011, New Zeala
- Tel Aviv University, Tel Aviv, Israel
- Haifa, Israel
- University of Alabama, United States
- École Polytechnique, Montreal, QC, Canada
- University of Sannio, UniversitÃ del Sannio di Benevento, Piazza Guerrazzi, 1, 82100 Benevento, Ita
- UC Irvine School of Humanities, University Dr, Irvine, CA 92697, USA
- Vienna University of Technology, Karlsplatz 13, 1040 Vienna, Austria
- University of Hull, Scarborough, North Yorkshire YO11, UK
- EPFL, 1015 Ecublens, Switzerland
- University of Nebraska-Lincoln, Lincoln, NE 68508, USA
- University City, Pennsylvania, USA
- Syracuse University, Syracuse, NY 13210, USA