Java Bytecode Instrumentation - Reconciling Developer Productivity

Aibek Sarimbekov, Yudi Zheng, Danilo Ansaloni, Walter Binder
University of Lugano, Lugano, Switzerland

Lubomir Bulej, Lukas Marek, Petr Tuma
Charles University, Prague, Czech Republic

Zhengwei Qi
Shanghai Jiao Tong University, Shanghai, China

March 23rd 2013
Dynamic Program Analysis Tools

Tools that observe relevant activities of running programs.

Examples:
- profiling
- debugging
- testing
- reverse engineering
- program comprehension
Instrumentation Classes vs. Analysis Classes
Instrumentation Classes vs. Analysis Classes

- Instrumentation Classes
- base program

at instrumentation time
Instrumentation Classes vs. Analysis Classes

base program

Analysis Classes

at run-time
Bytecode Instrumentation

- Javassist
- DiSL
- Soot
- ASM

- BCEL
- jChord
Bytecode Instrumentation

Which to use?
Can we raise the abstraction level for writing dynamic analysis tools, allowing software engineers to rapidly develop custom analysis tools, impairing neither expressiveness nor tool performance?
DiSL vs. ASM
Empirical Study

- Controlled Experiment
- Recasts of 10 open-source DPA Tools
DiSL: a DSL for Instrumentations

- Provide higher abstraction layer to write instrumentations
- Reduce development effort for writing instrumentations
- Avoid shortcomings of AOP-based approach
- Do not impair the performance of the resulting tools
DiSL at a Glance

Language constructs
- markers and snippets
- static and dynamic context
- scope and guards
- synthetic and thread local variables
DiSL Markers and Snippets Example

```java
public class DiSL {

    @Before(marker = BasicBlockMarker.class)
    public static void onBB() {
        Profile.profileBB(); // count number of executed basic blocks of code
    }

    @AfterReturning(marker = BytecodeMarker.class, args = "new")
    public static void onAlloc() {
        Profile.profileAlloc(); // count the number of allocated objects
    }
}
```
public class DiSL {

    @Before(marker = BasicBlockMarker.class)
    public static void onBB(MethodStaticContext msc,
        UniqueMethodId uid,
        BasicBlockStaticContext bbsc) {
        Profile.profileBB(  
            msc.thisMethodFullName(), // full method name  
            uid.get(),             // unique method ID (int value)  
            bbsc.getBBIndex(),     // basic block index (int value)  
            bbsc.getBBSize()       // bytecodes in the BB (int value)  
        );
    }
}
public class DiSL {

    @AfterReturning(marker = BytecodeMarker.class, args = "new")
    public static void onBB(DynamicContext dc) {
        // access allocated object
        Object allocObj = dc.getStackValue(0, Object.class);
        Profile.profileAlloc(allocObj);
    }
}
DiSL Scope and Guards Example

```java
public class DiSL {

    @Before(marker = BasicBlockMarker.class,
            scope = "TargetClass.*", // constrain instrumentation
            guard = LoopGuard.class) // constrain instrumentation
    public static void onBB(BasicBlockStaticContext bbsc) {
        Profile.profileBB(bbsc.getBBSIZE());
    }
}

public class LoopGuard {
    @GuardMethod
    public static boolean isApplicable(BasicBlockStaticContext bbsc) {
        return bbsc isFirstOfLoop(); // instrument only first BBs of loops
    }
}
```
public class DiSL {

    @SyntheticLocal
    static int bbsSL;
    @SyntheticLocal
    static long sizeSL;

    @Before(marker = BasicBlockMarker.class)
    public static void onBB(BasicBlockStaticContext bbsc) {
        bbsSL++;
        sizeSL += bbsc.getBBSize();
    }

    @After(marker = BodyMarker.class)
    public static void onMethodCompletion() {
        Profile.profileExecBytecodes(bbsSL, sizeSL);
    }
}
public class DiSL {

    @ThreadLocal
    static Profile profileTL;

    @Before(marker = BodyMarker.class, order = 1)
    public static void onMethodEntry() {
        if (profileTL == null) {
            profileTL = new Profile();
        }
    }

    @Before(marker = BasicBlockMarker.class, order = 0)
    public static void onBB(BasicBlockStaticContext bbsc) {
        profileTL.profileBB(bbsc.getBBSize());
    }
}
Empirical Study

- Controlled Experiment
- Recasts of 10 open-source DPA Tools
1. Goal: to identify a framework that allows rapid development of correct DPA tools

2. Subjects: students from Shanghai Jiao Tong University

3. Variables
   - independent: use of ASM or DiSL
   - dependent: time, correctness

4. Tasks: 6 typical instrumentations
Controlled Experiment: Procedure

- Self-assessment Questionnaire
- Tutorial on DPA
- Distribution of tasks
- Q&A Session
- Experiment
- Debriefing Questionnaire
Thank you for participating in our experiment. We very much appreciate your help. Please fill in the following questionnaire.

**Participant's details**
1. Full name: ______________________________________
2. Contact e-mail address: ____________________________
3. Age: ______________
4. Gender: ☐ male ☐ female
5. Affiliation: _______________________________________
6. Current highest academic degree:
   - ☐ Bachelor student
   - ☐ Master student
   - ☐ Ph.D. student
   - ☐ Professor
   - ☐ Other: ______________
7. Experience level in:
   - ☐ OOP
   - ☐ Java
   - ☐ Using Eclipse IDE
   - ☐ Dynamic Program Analysis
   - ☐ AspectJ
   - ☐ ASM
   - ☐ Linux
   - ☐ JVM and Java bytecode

**Task 2**
Implement the instrumentation that injects a call to `onArrayAllocation(int arrayLength)`, before each array allocation, where `arrayLength` is the length of the allocated array.

Note: your instrumentation should intercept only these bytecodes: NEWARRAY, ANEWARRAY

**Time needed for task completion:** ______ minutes

**Time Pressure:**
- ☐ Too much time pressure.
- ☐ Fair amount of pressure.
- ☐ Not so much time pressure.
- ☐ Very little time pressure.
- ☐ No time pressure at all.

**Difficulty:**
- ☐ trivial
- ☐ simple
- ☐ intermediate
- ☐ difficult
- ☐ impossible

**Comments:**
Controlled Experiment: Procedure

In total: 16 BSc, MSc, and PhD students
Controlled Experiment: Results

Time spent [minutes]

Correctness [points]

ASM

DISL

ASM

DISL
Controlled Experiment: Results

<table>
<thead>
<tr>
<th></th>
<th>Time [minutes]</th>
<th>Correctness [points]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASM</td>
<td>DiSL</td>
</tr>
<tr>
<td><strong>Summary statistics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>148.62</td>
<td>54.62</td>
</tr>
<tr>
<td>difference</td>
<td>-63.2%</td>
<td>+46.6%</td>
</tr>
<tr>
<td>min</td>
<td>130</td>
<td>17</td>
</tr>
<tr>
<td>max</td>
<td>180</td>
<td>140</td>
</tr>
<tr>
<td>median</td>
<td>145</td>
<td>45.5</td>
</tr>
<tr>
<td>stdev.</td>
<td>18.73</td>
<td>38.96</td>
</tr>
</tbody>
</table>

| **Assumption checks** | | | |
| Kolmogorov-Smirnov Z | 0.267 | 0.203 | 0.241 | 0.396 |
| Levene F             | 1.291 |     | 1.939 |     |

| **One-tailed Student’s t-test** | | |
| df | 14 | 14 |
| t  | 6.150 | -3.746 |
| p-value | <0.001 | 0.002 |

Descriptive statistics
DiSL improves developer productivity compared to ASM. Results are both practically and statistically significant.
Empirical Study

✦ Controlled Experiment
✦ Recasts of 10 open-source DPA Tools
Recasts

10 open-source DPA tools.

Metrics:

- Logical SLOC
- Performance Overhead
Logical SLOC

![Graph showing Logical SLOC with various software tools like Senseo, RacerAJ, JCarder, Cobertura, TamiFlex, JRrat, and Emma.]
Performance Overhead

- benchmarks from the Dacapo-9.12 suite
- geometric mean

Four quad-core Intel Xeon CPUs E7340, 2.4 GHz, 16 GB RAM, Ubuntu GNU/Linux 11.04 64-bit with kernel 2.6.38, Oracle Java HotSpot 64-bit Server VM 1.6.0_29
Performance Overhead

steady-state

Four quad-core Intel Xeon CPUs E7340, 2.4 GHz, 16 GB RAM, Ubuntu GNU/Linux 11.04 64-bit with kernel 2.6.38, Oracle Java HotSpot 64-bit Server VM 1.6.0_29
Conclusions

RQ: Can we raise the abstraction level for writing dynamic analysis tools, allowing software engineers to rapidly develop custom analysis tools, impairing neither expressiveness nor tool performance?
Conclusions

RQ: Can we raise the abstraction level for writing dynamic analysis tools, allowing software engineers to **rapidly** develop custom analysis tools, impairing neither expressiveness nor tool performance?

A: yes, use DiSL!