Using AOP for Detailed Runtime Monitoring Instrumentation

Jonathan E Cook, joncook@nmsu.edu
Amjad Nusayr,anusayr@cs.nmsu.edu

The 2009 Workshop on Dynamic Analysis

New Mexico State University
Runtime Monitoring

- The act of observing an executing system in order to learn something about its dynamic behavior
- RM needs an extremely wide variety of instrumentation mechanisms
Aspect Oriented Programming

- An elegant framework for constructing program behaviour that is orthogonal to the underlying program code base

- AOP is a natural fit for the domain of runtime monitoring
AOP Weaving vs Runtime monitoring instrumentation

- Code to be instrumented
- Advice to be weaved

Underlying program
Aspect Oriented Programming

- **Weaving**: the process of instrumentation
- **Advice**: code that will be weaved
- **Jointpoint**: points in the program where advice can be weaved
  - method call, object construction
- **Aspect**: an entity that holds all of the above
AOP for Runtime Monitoring

- Naturally captures the idea of scattered instrumentation in a base program
- Can be used on existing programs
- It is formal and uses normal programming concepts that programmers can readily grasp
AOP Deficiencies

- Not enough detail to cover all runtime monitoring needs
  - e.g., statement level weaving, basic blocks, loops, local variable access
- Limited to weaving based on the source code
  - Sampling-based profiling needs weaving based on execution time intervals rather than on places in the code
final double matgen(double a[][], final int n, double b[]) {
    ....
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            init = 3125 * init % 65536;
            a[j][i] = (init - 32768.0) / 16384.0;
            norma = (a[j][i] > norma) ? a[j][i] : norma;
        }
    }
    for (j = kp1; j < n; j++) {
        col_j = a[j];
        if (l != k) {
            col_j[l] = col_j[k];
            col_j[k] = t;
        }
        daxpy(n - (kp1), t, col_k, kp1, 1, col_j, kp1, 1);
    }
    ....
}
final double matgen(double a[][], final int n, double b[]) {
    ....
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            init = 3125 * init % 65536;
            a[j][i] = (init - 32768.0) / 16384.0;
            norma = (a[j][i] > norma) ? a[j][i] : norma;
        }
    }
    ....
    for (j = kp1; j < n; j++) {
        col_j = a[j];
        if (l != k) {
            col_j[l] = col_j[k];
            col_j[k] = t;
        }
        daxpy(n - (kp1), t, col_k, kp1, 1, col_j, kp1, 1);
    }
    ....
}
Axes of Weaving

Weaving in code and data space

Current AOP coverage

Weaving in two dimensions
Extending the axes of weaving to a 3 dimensional view

Axes of Weaving

Extended AOP coverage

Data space

Time

Code space
Axes of Weaving

- Time
- Data space
- Code space
- Sampling

extended AOP coverage
New Code PCDs

- An extension in abc (AspectJ)
- New *basicblock* pointcut designator enables advice on every basic block
- New *loopbackedge* pointcut designator enables advice on every loop
- Both give reflective information
  - Class and Method name (already existing)
  - In-method unique ID (additional)
aspect TraceBasicBlocks {
    before(int blockID) : basicblock() && args(blockID) {
        System.err.println("Entering Block --> " + blockID + " at" + thisJoinPoint.getSourceLocation());
    }
    after(int blockID) : basicblock() && args(blockID) {
        System.err.println("Exiting Block --> " + blockID);
    }
}
aspect TraceLoops {
    before(int id) : loopbackedge() && args(id) {
        System.err.println("Loop body done, " + id + " at " +
            thisJoinPoint.getSourceLocation());
    }
}
AOP / RM Issues

- ABC was specifically created for extensibility, but is still limited
  - When we tried statement-level advice, we were told “we never intended abc for that!”
- For RM, we implement before and after advice, but not around advice
  - Would around be useful?
AOP / RM Issues

- ABC weaving occurs on an intermediate representation
  - e.g., all loops translated to if-goto structures
  - can we ensure source code fidelity?

- After advice misses final logical compare
  - single JVM compare-branch instruction
  - can be fixed with code duplication
AOP / RM Issues

- Ultimate goal: performance
  - abc implements advice as method call
  - can we rely on optimizing JVMs?
Examples

- Benchmark suite
  - JTetris: Tetris game in Java
  - Image2Html: converts a bitmap image into HTML
  - Java Linpack, an implementation in Java of the FORTRAN Linpack routines

- Coverage analysis.
  - Full instrumentation and Key class instrumentation

- Profiling
  - Time
  - Probability
## Results

<table>
<thead>
<tr>
<th>Application</th>
<th>Total number of blocks</th>
<th>Number of methods and loops</th>
<th>Time no Instrumentation</th>
<th>Prob= .5</th>
<th>Prob= .5</th>
<th>Prob= .05</th>
<th>Prob= .05</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Time Block Instr</td>
<td>- Time Loop Instr</td>
<td>- Time Block Instr</td>
<td>- Time Loop Instr</td>
</tr>
<tr>
<td>Java linpack</td>
<td>156</td>
<td>38</td>
<td>0.0675</td>
<td>0.572</td>
<td>0.335</td>
<td>0.271</td>
<td>0.187</td>
</tr>
<tr>
<td>J-Tetris</td>
<td>240</td>
<td>84</td>
<td>0.3275</td>
<td>0.547</td>
<td>0.435</td>
<td>0.439</td>
<td>0.339</td>
</tr>
<tr>
<td>Image2Html</td>
<td>409</td>
<td>39</td>
<td>0.6611</td>
<td>2.311</td>
<td>0.819</td>
<td>0.967</td>
<td>0.735</td>
</tr>
</tbody>
</table>
Future work.

- Continue to work new joinpoint types
  - loop body, if-else body, case body
  - time and probability dimensions
- Design, prototype, implement, test, and evaluate new pointcuts in the new dimensions
- Mechanisms for making reflective information easier and faster to obtain in the advice code will be needed
Thank you

Questions ?
Sampling based profiling

Weave on event

Weave on event

Weave on event

time