Identification of Differences Between Aspect-Oriented Programs

Marija Katic, PhD Student

University of Zagreb, Faculty of Electrical Engineering and Computing
Department of Applied Computing
Content

Aspect-Oriented Programming

Motivation for differencing…

Representation of Aspect-Oriented Programs

Representation with Control-Flow Graphs

Aspect-Oriented Control Flow Graph

Differencing Algorithm for Aspect-Oriented Programs

Hammock Graphs

Example of Difference Identification

Results
Aspect-Oriented Programming
Problem of cross-cutting concerns

Logging, transaction management, data persistence, exception handling ...

Lower software quality (code redundancy, maintenance, ...)

Aspect-oriented programming

not replacement, but complement to object-oriented paradigm

Aspect-oriented program consists of two parts:

Base code - more purpose-specific; classes

Aspect code - cross-cutting code; aspects, advice
Aspect Weaving - process of merging two program code parts

Static - weaving aspects into classes at compile-time

Dynamic - weaving aspects into classes at run-time

Advice - before, after, around

Join-Point - point in the program where advice is woven

Pointcut - specify rules for advice weaving

Representation of AspectJ programs

Aspect-oriented extension to Java

Static weaving
## Aspect-Oriented Programming (3/3)

### Join-Point Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Join-Point Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method call</td>
<td>call join-point</td>
</tr>
<tr>
<td>Method execution</td>
<td>execution join-point</td>
</tr>
<tr>
<td>Constructor call</td>
<td></td>
</tr>
<tr>
<td>Constructor execution</td>
<td></td>
</tr>
<tr>
<td>Field read access</td>
<td></td>
</tr>
<tr>
<td>Field write access</td>
<td></td>
</tr>
<tr>
<td>Advice execution</td>
<td></td>
</tr>
<tr>
<td>Exception handler execution</td>
<td></td>
</tr>
</tbody>
</table>
Motivation
Need for Differencing of Aspect-Oriented Programs

Impact analysis, regression testing, generation of program updates, ...

Classification of program constructs
added, deleted, modified, unchanged

Changes in aspect-oriented programs
addition, deletion, modification - class, interface, method, field, aspect, pointcut, advice
Representation of Aspect-Oriented Programs
Choosing an Appropriate Representation (1/2)

Some of well-known existing representations for AOP:


before and after advice only, around advice excluded complex concepts such as multiple advice weaving into the same join-point and dynamic advice are not considered


accounts for complex concept of around advice, multiple advice application and dynamic advice represents the inter-procedural interactions in the program
Choosing an Appropriate Representation (2/2)

Good representation for OOP:


Identification of changes specific to object-oriented behaviour

Let us adapt it for AOP …
Control-Flow Graph (CFG) of a method $m\ CFG_m$

directed graph that represents all paths that might be traversed through a method during its execution

$CFG_m = (N, E, n_s, n_e)$

$N$ – set of nodes (module statements)

$E$ – set of edges (flow of control)

$n_s$ – single entry node

$n_e$ – single exit node

Aspect-Oriented Control-Flow Grap - new CFG representation of an aspect-oriented program (Aspect Graph, Around Graph)
Aspect Graph (1/2)

$n_p$ – node for a join-point statement $p$

**Aspect Graph** (AG) for $n_p$ AG$_{np}$

control-flow graph that represents paths of execution for the join-point $p$ with woven advice

AG$_{np} = (N_{np}, E_{np}, aentry_{np}, aexit_{np})$

$N_{np}$ – set of nodes (advice and join-point statements)

$E_{np}$ – set of edges (flow of control)

$aentry_{np}$ – entry node of AG (aspect code entry node)

$aexit_{np}$ – exit node of AG (aspect code exit node)
Node in CFG for a join-point statement $p$; there is a set of advice that applies to it.
Around advice! – complicates the representation a bit!

**Around Graph** (ARNG) for around advice $a \text{ARNG}_a$

control-flow graph that represents paths of execution for around advice (including nested advice and the join-point)

$$\text{ARNG}_a = (N_a, E_a, \text{arnentry}_a, \text{arnexit}_a)$$

- $N_a$ – set of nodes (advice and join-point statements)
- $E_a$ – set of edges (flow of control)
- $\text{arnentry}_a$ – entry node of ARNG (around entry node)
- $\text{arnexit}_a$ – exit node of ARNG (around exit node)

ARNG is a sub-graph of aspect graph
**Aspect-Oriented Control-Flow Graph (AO-CFG)**

(AO-CFG) of a method \( m \) \( \text{AOCFG}_m \)

extension of \( CFG_m = (N, E, n_s, n_e) \) for \( m \); represents all execution paths within \( m \)

**Idea for AO-CFG construction:**

replace node for **join-point statement** \( p_i \), \( i = 1, \ldots, n \) with **aspect graph**:

\[
\text{node } n_{pi} \text{ is replaced with } AG_{npi} = (N_{npi}, E_{npi}, aentry_{npi}, aexit_{npi})
\]

**AOCFG** \( m \) = \( (N', E', n_s', n_e') \)

\[
N' = \bigcup_{i=1}^{n} N_{pi} \cup N \setminus \bigcup_{i=1}^{n} P_i
\]

\[
E' = \bigcup_{i=1}^{n} E_{pi} \cup E
\]

\[
n_s' = n_s; n_e' = n_e
\]
Aspect-Oriented Control-Flow Graph (2/6)
Aspect-Oriented Control-Flow Graph (3/6)

Join-Point Representation

Representation of call and execution join-points

Difference between call and execution join-points:

- Call join-point
- Execution join-point

advice application
Call join-point representation

same as Apiwattanapong et al.

accounts for dynamic binding
Aspect-Oriented Control-Flow Graph (5/6)

Call Join-Point Representation

Aspect Graph for call join-point
Aspect Graph for execution join-point

entire method is a join-point
Differencing Algorithm for Aspect-Oriented Programs
Notion of **Hammock Graphs** is used!

**Hammock** \( H = (N, E, n_s, n_e) \)

sub-graph of a control-flow graph \( CFG \) with *entry node* \( n_s' \) in \( H \), and *exit node* \( n_e' \) not in \( H \) such that:

- all edges from \( (CFG \setminus H) \) to \( H \) go to \( n_s' \)
- all edges from \( H \) to \( (CFG \setminus H) \) go to \( n_e' \)

\( H \sim \) **minimal** if there is no hammock \( H' \) with the entry node \( n_s \) with fewer number of nodes
public int fibonacci(int m) {
    int f0 = 0, f1 = 1, f2 = 0;
    if (m<=1) {
        return m;
    } else {
        for(int i= 2; i<= m; i++) {
            f2 = f0 + f1;
            f0 = f1;
            f1 = f2;
        }
        return f2;
    }
}
Aspect Hammock

**Aspect Hammock** $G = (N, E, u, v)$

- corresponds to the aspect graph $G' = (N', E', u', v')$ with the exception of exit node $v'$

- $u \in N$ corresponds to the *aspect code entry* node $u'$

- $v \notin N$ corresponds to a successor of the *aspect code exit* node $v'$

**Execution Aspect Hammock** - join-point is of an execution type
Around Hammock $G = (N, E, u, v)$

corresponds to the around graph $G' = (N', E', u', v')$ with the exception of the exit node $v'$

$u \in N$ corresponds to the around advice entry node $u'$

$v \notin N$ corresponds to a successor of the around advice exit node $v'$
Join-Point Hammock

**Join-Point Hammock** $G = (N, E, u, v)$

- sub-hammock of the aspect hammock
- $u \in N$ corresponds to the entry node of the join-point
- $v \notin N$ corresponds to the successor of the join-point
aspect hammock

around hammock

join-point hammock
Extension of the differencing algorithm for object-oriented programs

Algorithm CalcDiffAO

1. compare classes, interfaces and aspects; add matched pairs to sets C, I and A

2. for each \((a, a')\) in A do
   3. compare advice; add matched pairs to ADV

4. for each \((adv, adv')\) in ADV do
   5. create CFGs \(G\) and \(G'\)
   6. create hammocks in \(G\) and \(G'\) until one node \(na\) and \(na'\) left
   7. \(NA \leftarrow NA \cup \text{HmMatch}(na, na', \lambda, \sigma)\)

8. for each \((c, c')\) in C, I or A do
   9. compare modules; add matched pairs to M

10. for each \((m, m')\) in M do
   11. create AO-CFGs \(AOCFG\) and \(AOCFG'\) for modules \(m\) and \(m'\)
   12. create hammocks in \(AOCFG\) and \(AOCFG'\) until one node \(n\) and \(n'\) left
   13. \(N \leftarrow N \cup \text{HmMatchAO}(n, n', \lambda, \sigma, NA)\)
Differencing Algorithm for Aspect-Oriented Programs (3/4)

Algorithm \texttt{HmMatchAO}

1. if at least one element in pair of nodes (n, n’) is aspect hammock then
2. classify advice into sets with added, deleted, modified and unchanged advice
3. identify join-point nodes in (n, n’)
4. compare join-points with \texttt{HmMatch}
Differencing Algorithm for Aspect-Oriented Programs (4/4)
## Example of Difference Identification (1/6)

### BASE CODE

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><code>int fact(int n) {</code></td>
<td><code>int fact(int n) {</code></td>
</tr>
<tr>
<td>2.</td>
<td><code>if (n &lt;= 1)</code></td>
<td><code>if (n &lt;= 1)</code></td>
</tr>
<tr>
<td>3.</td>
<td><code>return 1;</code></td>
<td><code>return 1;</code></td>
</tr>
<tr>
<td>4.</td>
<td><code>else</code></td>
<td><code>else</code></td>
</tr>
<tr>
<td>5.</td>
<td><code>return n*fact(n-1);</code></td>
<td><code>return n*fact(n-1);</code></td>
</tr>
<tr>
<td>6.</td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

### ASPECT CODE

<table>
<thead>
<tr>
<th></th>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>pointcut executionMethod() :</strong></td>
<td><strong>pointcut executionMethod() :</strong></td>
</tr>
<tr>
<td></td>
<td><code>execution(int fact(int));</code></td>
<td><code>execution(int fact(int));</code></td>
</tr>
<tr>
<td></td>
<td><strong>before() :</strong> executionMethod() {</td>
<td><strong>before() :</strong> executionMethod() {</td>
</tr>
<tr>
<td></td>
<td><code>print(&quot;entry:&quot;);</code></td>
<td><code>print(&quot;entry:&quot;);</code></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td><strong>int around() :</strong> executionMethod() {</td>
<td><strong>before() :</strong> executionMethod() {</td>
</tr>
<tr>
<td></td>
<td><code>print(&quot;around:&quot;);</code></td>
<td><code>print(&quot;entry:&quot;);</code></td>
</tr>
<tr>
<td></td>
<td><code>return proceed();</code></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td><strong>after() :</strong> callMethod() {</td>
</tr>
<tr>
<td></td>
<td><strong>after() :</strong> executionMethod() {</td>
<td><code>print(&quot;exit:&quot;);</code></td>
</tr>
<tr>
<td></td>
<td><code>print(&quot;exit:&quot;);</code></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>
Example of Difference Identification (3/6)
Example of Difference Identification (4/6)
Example of Difference Identification (5/6)
Example of Difference Identification (6/6)
### Results (1/2)

<table>
<thead>
<tr>
<th>Experimented program</th>
<th>LOC</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>678</td>
<td>3669</td>
<td>3718</td>
</tr>
<tr>
<td></td>
<td>680</td>
<td>3079</td>
<td>3128</td>
</tr>
<tr>
<td></td>
<td>1066</td>
<td>5867</td>
<td>5984</td>
</tr>
<tr>
<td>Spacewar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1675</td>
<td>4495</td>
<td>4725</td>
</tr>
<tr>
<td></td>
<td>1784</td>
<td>6213</td>
<td>6454</td>
</tr>
<tr>
<td></td>
<td>1711</td>
<td>4576</td>
<td>4809</td>
</tr>
<tr>
<td>HealthWatcher</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5800</td>
<td>16525</td>
<td>17212</td>
</tr>
<tr>
<td></td>
<td>5924</td>
<td>17337</td>
<td>18113</td>
</tr>
<tr>
<td></td>
<td>5850</td>
<td>16399</td>
<td>17137</td>
</tr>
<tr>
<td></td>
<td>5977</td>
<td>17439</td>
<td>18225</td>
</tr>
<tr>
<td></td>
<td>5983</td>
<td>17461</td>
<td>18247</td>
</tr>
<tr>
<td>AJHotDraw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>23086</td>
<td>67814</td>
<td>76790</td>
</tr>
<tr>
<td></td>
<td>22834</td>
<td>66955</td>
<td>75838</td>
</tr>
<tr>
<td></td>
<td>23004</td>
<td>67573</td>
<td>76537</td>
</tr>
<tr>
<td>AJHSQILDB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>70167</td>
<td>205421</td>
<td>223404</td>
</tr>
</tbody>
</table>
Thank you!

Questions?