Advice weaving

Ganesh Sittampalam
Overview

• Match - produce mapping:
  application sites ➞ advice + dynamic residue

• Prepare application sites

• Weave “inside-out” (i.e. in reverse precedence order)
Pointcut separation

- Restrict containing class
  - e.g. within(…)
  - Does include nested classes
- Restrict containing method
  - e.g. withinCode(…)
  - Doesn’t include classes lexically within the method
- Specific join point
  - e.g. call(…)

Restrict containing class
Translating pointcuts

```java
execution(int Foo.foo(char))
   => withinmethod(int Foo.foo(char)) && execution()

execution(Foo.new(int))
   => withinconstructor(Foo.new(int)) && execution()

adviceexecution() => withinadvice() && execution()

staticinitialization(Foo)
   => within(Foo) && withinstaticinitialization() && execution()

preinitialization(Foo.new(int))
   => withinconstructor(Foo.new(int)) && preinitialization()

call(int Foo.foo(char)) => methodcall(int Foo.foo(char))
call(Foo.new(int)) => constructorcall(Foo.new(int))
```
Initialization

initialization(Foo.new(int))
\[
\Rightarrow \text{withinconstructor}(\text{Foo.new(int)}) \\
&\& \text{classinitialization}()
\]

initialization(Foo.new())
\[
\Rightarrow (\text{withinconstructor}(\text{Foo.new()}) \\
&\& \text{classinitialization}()) \\
| | \text{interfaceinitialization}(\text{Foo})
\]

initialization(Foo.new(..))
\[
\Rightarrow (\text{withinconstructor}(\text{Foo.new(..)}) \\
&\& \text{classinitialization}()) \\
| | \text{interfaceinitialization}(\text{Foo})
\]
Pointcut preprocessing

• Inline named pointcuts
  – requires “private” pointcut variables
    pointcut bar(int x) : args(x,..)  
    bar(*) ⇒ private(int x) { args(x,..) }

• Convert to DNF
  – to correctly handle alternative bindings
    (this(x) || target(x)) && if(x instanceof Foo)  
    ⇒ (this(x) && if(...)) || (target(x) && if(...))

• Lift pointcuts from cflow and per clauses into special advice declarations
  – look for CSE and counter opportunities with cflow pointcuts
Restructuring

- **Move** `new+invokespecial` **together**
  - Needed for constructor call matching

- `foo()` ➔ `a0 = foo()`
  - If `foo()` returns a value we want to bind

- **Restructure** `return` statements in body so that there is just one at the end
  - For execution pointcuts

- **Inline** `this(...)` calls in constructors
  - For initialization and preinitialization weaving
Matching

• Shadows categorised as:
  – Whole body (execution, initialization etc)
  – Individual statement (method call, field set, field get etc)
  – Pair of statements (constructor call)
  – Exception handler

• Iterate through all weavable classes
  – At each shadow, try all pointcuts
Finding method call shadows

... if (stmt instanceof InvokeStmt) {
    InvokeStmt istmt=(InvokeStmt) stmt;
    invoke=istmt.getInvokeExpr();
} else if (stmt instanceof AssignStmt) {
    AssignStmt as = (AssignStmt) stmt;
    Value rhs = as.getRightOp();
    if (!(rhs instanceof InvokeExpr)) return null;
    invoke=(InvokeExpr) rhs;
} else return null;

SootMethodRef methodref=invoke.getMethodRef();
Dynamic residues

- Mini-language roughly corresponding to structure of pointcuts
- Used to generate runtime code
  - decide whether advice should execute
  - bind values to pass to advice
- Also used to signal static results
  - “Match failed”
  - “This always matches”
- Easy to improve residues using analysis results
Dynamic residue construction

• “pre” residue from aspect
  – hasAspect check for per advice
• Residue from pointcut
• Residue from advice spec (before, after etc)
• “post” residue from aspect
  – aspectOf for getting aspect instance
Weaving

• Insert nops around the instruction(s) representing the shadow
  – Take care to fix up exception ranges and gotos correctly
• Advice gets inserted just inside the nops
• Advice gets woven “inside-out”