SCoPE: an AspectJ Compiler for Supporting User-Defined Analysis-Based Pointcuts

Tomoyuki Aotani
Hidehiko Masuhara
http://www.graco.c.u-tokyo.ac.jp/ppp/index.php?Projects/scope

Motivation

- Fragile aspects
  - Pointcuts are using immediate properties (method signatures).
  - A small modification to the program could require changes in pointcuts.

Motivation

- Analysis-based pointcuts
  - Pointcuts use properties obtained from static program analyses.
  - Stable against program evolution.

Analysis-Based Pointcuts

- Class structure analysis
  - Pointcuts match join points based on class structural properties.
  - E.g. insert code at execution of constructor of classes that have a field foo of type Foo:

```
Class Bar_1 {
    Foo foo;
    public Bar_1() {
        …
        //insert new code here
    }
}
```

Analysis-Based Pointcuts

- AspectJ
  - Pointcut contain a list of classes that have field foo. (Object Bar_1 to Bar_n have field foo)

```
pointcut initObjectWithFooField() :
      execution(Bar_1.new(..)) || ...
      execution(Bar_n.new(..));

after() returning() : initObjectWithFooField() {
    //insert code
}
```

SCoPE (Static Conditional Pointcut Evaluation)

- A compilation scheme
  - Evaluate analysis-based pointcut at compile time
    - Using properties from static program analyses
    - Efficiency – no runtime overhead
  - User-defined pointcut
    - Wide-range of program analyses
  - Use existing conditional pointcut (if) syntax
    - No syntactic extensions

Analysis-Based Pointcuts

- SCoPE

```
pointcut initObjectWithFooField() : execution(*.new(..)) &&
    if(hasfield(thisJoinPoint, "foo");

static boolean hasfield(JoinPoint tjp, String fname) {
    try {
            tjp.getSignature().getDeclaringType().getField(fname);
            return true;
    } catch (Exception e) { return false; }
}
```

Conditional Pointcut (if)

- AspectJ Compiler
  - Compile conditional pointcut into runtime test
- SCoPE Compiler
  - Need to separate dynamic and static conditional pointcut

Definition of Static Conditional Pointcuts

- A conditional pointcut is static if its expression always returns the same value with respect to the same join point shadow.

Definition of Static Conditional Pointcuts

- if(e) is static if:
  - e is a immutable class variable

    final static boolean flag = true;

    pointcut isActive() : if(flag);

  - e does not have a variable bound by other pointcuts.

    pointcut nullArgument(Object x) : args(x) && if(x == null);

Definition of Static Conditional Pointcuts

- if(e) is static if:
  - e is not an invocation of a dynamic method

    final static Map map = new HashMap();

    pointcut mapMethod() : if(map.get(…).boolValue())

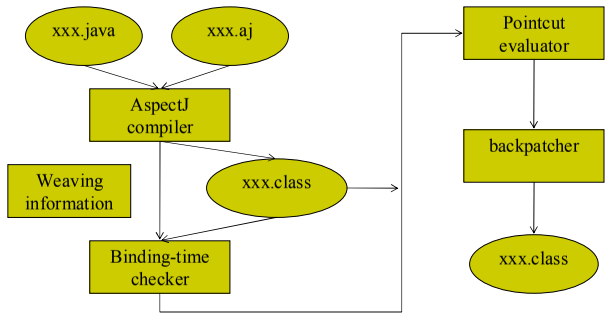
  - e is not an predetermined dynamic method

    java.lang.Class.newInstance()

    org.aspectj.lang.JoinPoint.getArgs()

    etc…

SCoPE Compilation Scheme



SCoPE Compilation Scheme

```
Class Bar_1 {
    Foo foo;
    public Bar_1() {
        …
    }
}

Class Bar_2 {

    public Bar_2() {
        …
    }
}

aspect StructureAnalysis {
    pointcut initObjectWithFooField() execution(*.new(..)) &&
    if(hasfield(thisJoinPoint, "foo");

    static boolean hasfield(JoinPoint tjp, String fname)  {…}
    after() returning() : initObjectWithFooField() {…}
}
```

SCoPE Compilation Scheme

- AspectJ compiler
  - Generate weaving information
  - Conditional pointcuts are compiled into if-residues

shadow location	if-residue name	parameters
1.5	if$1	shadow$1
2.4	if$1	shadow$2

SCoPE Compilation Scheme

```
Class Bar_1 {
    final static JoinPoint.StaticPart shadow$1 = …;

    Foo foo;
    public Bar_1() {
        …
        if (StructureAnalysis.if$1(shadow$1)) {
            StructureAnalysis.aspectOf().after$1();
        }
    }
}

Class Bar_2 {
    final static JoinPoint.StaticPart shadow$2 = …;

    public Bar_2() {
        …
        if (StructureAnalysis.if$1(shadow$2)) {
            StructureAnalysis.aspectOf().after$1();
        }
    }
}
```

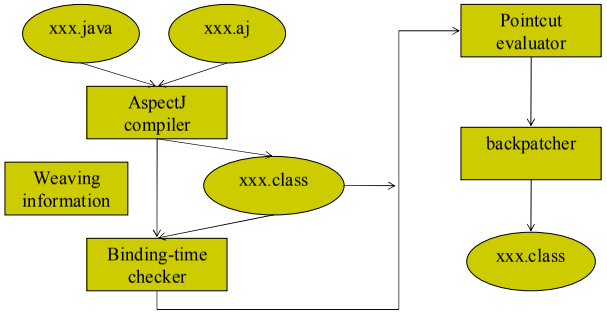
SCoPE Compilation Scheme

```
class StructureAnalysis {
    //if-residue
    //pointcut
    static boolean if$1(…) {
        return hasfield(…);
    }

    static boolean hasfield(…)  {…}

    //advice
    void after$1() {…}
}
```

SCoPE Compilation Scheme



SCoPE Compilation Scheme

- Binding-time checker
  - For each if-residue, decide whether it is static using the definitions.
- Pointcut Evaluator
  - Evaluate static if-residues
  - if$1(shadow$1) = true
  - if$1(shadow$2) = false

SCoPE Compilation Scheme

- Backpatcher
  - Replace each static if-residue with constant instruction.
  - if(true) //shadow$1
  - if(false) //shadow$2

SCoPE Compilation Scheme

```
Class Bar_1 {
        Foo foo;
        public Bar_1() {
                …
                if (true) {
                    //inserted code
                }
        }
}

Class Bar_2 {
        public Bar_2() {
                …
                if (false) {
                    //inserted code
                }
        }
}
```

Performance

- Compared to AspectBench Compiler (abc)
- Compilation time
  - Similar speed as abc if pointcut evaluation step is excluded
  - Pointcut evaluation step mainly runs user-defined analysis methods

Performance

- Runtime Overheads
  - Split single run into the initialization phase and three consecutive iterations.
  - Measure time spent on each phase.
  - Only 1st iterations have significant overhead.
    - Due to conditional branch with a constant value
    - JVM first executes a program by using a bytecode interpreter.
    - Will cause overhead until dynamic compiler optimizes it.

Conclusion

- Small amount of overhead
- User-defined – Flexibility
- No additional syntax

Also Fights Gingivitis