

Aspect-Oriented Programming

G. Kiczales, J. Lamping,
A. Mendhekar, C. Maeda, C. Lopes,
J. Loingtier, J. Irwin

Outline

- Motivation
- Basics of AOP
- AspectJ
- Example
- Assessment
- Conclusions
- Discussion

2

Motivation for AOP

- Before AOP, we had:
 - Procedural approach
 - Object-oriented programming approach
- Problem: There are some programming problems that fit neither the procedural approach nor the OOP approach.
- Solution: Aspect-oriented programming

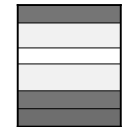
3

Motivation for AOP

- Image processing example
 - First implementation was easy to develop and maintain, but made poor use of memory
 - Optimized version was “tangled”
- Cross-cutting concerns
 - Tangling
 - Scattering



First implementation



Optimized implementation

4

Basics of AOP

- Aspect-oriented programming modularizes cross-cutting concerns
- Components
 - Encapsulated by GP languages
 - Functional units
- Aspects
 - Not encapsulated by GP languages
 - Non-functional units
 - Cross-cut

5

Basics of AOP

- Component language
- Component program
- Aspect language
- Aspect program
- Aspect weaver

6

Basics of AOP

- Join point: well-defined point in the execution of a component program
- Pointcut: collection of join points; a way to pick out join points
 - Defined in the aspect program
- Advice: code that executes at each join point picked out by a pointcut
 - Defined in the aspect program

7

AspectJ

- An extension to Java that adds AOP capabilities
- Developed at Xerox PARC
- Let's look at an example...

8

```
public class TestClass
{
    public void sayHello ()
    {
        System.out.println ("Hello!");
    }

    public void sayAnything (String s)
    {
        System.out.println (s);
    }

    public static void main (String[] args)
    {
        sayHello ();
        sayAnything ("ok.");
    }
}
```

9

Program Output

```
$ javac TestClass.java
$ java TestClass
Hello!
ok.
$
```

10

```
public aspect MyAspect
{
    public pointcut sayMethodCall (): call (public void
        TestClass.say*( ) );
    public pointcut sayMethodCallArg (String str): call
        (public void TestClass.sayAnything (String)
        && args(str);

    before(): sayMethodCall() {
        System.out.println("\n TestClass." +
            thisJoinPointStaticPart.getSignature().getName() +
            " start...");
    }
    after(): sayMethodCall() {
        System.out.println("\n TestClass." +
            thisJoinPointStaticPart.getSignature().getName() +
            " end...");
    }
    before(String str): sayMethodCallArg(str) {
        if (str.length() < 3) {
            System.out.println ("Error: I can't say
                words less than 3 characters");
            return;
        }
    }
}
```

11

```
public class TestClass
{
    public void sayHello ()
    {
        System.out.println ("Hello!");
    }

    public void sayAnything (String s)
    {
        System.out.println (s);
    }

    public static void main (String[] args)
    {
        sayHello ();
        sayAnything ("ok.");
    }
}
```

12

AspectJ Output

```
$ ajc MyAspect.aj
  TestClass.java
$ java TestClass
TestClass.sayHello start...
Hello!

TestClass.sayHello end...
Error: I can't say words less
      than 3 characters.
ok.
$
```

13

Assessment

- Reduction in bloat due to tangling metric
$$\frac{\text{tangled code size} - \text{component code size}}{\text{sum of aspect program sizes}}$$
- Language support for separation of concerns and clean abstraction and composition of components and aspects
- Adam Kolawa – “AOP is not the best strategy to deal with problems it attempts to solve.”

14

Conclusions

- AOP addresses the limitations of current programming paradigms
- Component vs. Aspect
- AOP is a young idea. The authors think it is the next step in the evolution of programming.

15

Discussion

- What types of aspects might be useful?
- Are there drawbacks to AOP? Is Adam Kolawa right?
- Is AOP as big a shift as OOP was?

16

References

- G.Kiczales, et al. An Overview of AspectJ.
- R. Walker, E. Baniassad, G. Murphy. An Initial Assessment of AOP.
- A. Kolawa. Ghost from the Past.
- J. Haddock-Schatz. Aspect-Oriented Programming.

17