Introduction to Aspect-Oriented Software Development

Bruno Harbulot
ESNW, the University of Manchester
http://www.cs.man.ac.uk/~harbulob/

ELF Developers' Forum - Manchester - October 2005
Presentation outline

- Problem: code-tangling
- Concepts of Aspect-Oriented Programming
- AOP Tools
  - AspectJ
- Applications: Design pattern example
- Conclusions
Problem presentation
**good modularity**

URL pattern matching in org.apache.tomcat
- red shows relevant lines of code
- nicely fits in two boxes (using inheritance)

Source: http://www.eclipse.org/aspectj/teaching.php

(c) Copyright 1998-2002 Palo Alto Research Center Incorporated. All rights reserved.
problems like...

- logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

Source: http://www.eclipse.org/aspectj/teaching.php

(c) Copyright 1998-2002 Palo Alto Research Center Incorporated. All rights reserved.
Concepts of Aspect-Oriented Programming
Crosscutting concerns

- **Concern**: “specific requirement or consideration that must be addressed in order to satisfy the overall system goal” [Lad03]
- Designing software: separating concerns into units such as procedures, classes, methods, libraries, etc.
- Two concerns *crosscut* each other when their relation implies tangled code.
- **Crosscutting concern**: concern that crosscuts the main purpose of a unit, or that is spanned across multiple units.
Motivation for Aspect-Oriented Programming

- Programming paradigm for encapsulating crosscutting concerns.
- AOP builds on top of other programming paradigms: object-oriented, procedural or functional. It does not supplant them.
- Encapsulate crosscutting concerns into aspects.
Concepts of AOP (I)

- **Aspect**: unit encapsulating a crosscutting concern.
- **Join point**: point in the execution of a program where an aspect might intervene.
- “[…] whenever condition C arises, perform action A” [Fil05]
  - **Pointcut**: expression of a subset of join points (condition C)
  - **Advice**: piece of code for action A.
  - Pointcuts and advice encapsulated into aspects.
Concepts of AOP (II)

- AOP is not about “patching” pieces of code.

- AOP is about performing an action systematically upon recognition of a behaviour in the code.
Concepts of AOP (III)

- Weaving

Diagram:
- Base units
- Aspects
- Weaver
- Application
Aspect-Oriented Programming Tools
AOP Tools

• (Java-based tools)

• Languages:
  – AspectJ (now merged with AspectWerkz)

• Frameworks (mainly for J2EE applications):
  – JAC
  – Jboss AOP
AspectJ

- Aspect-Oriented extension to Java.
- Aspect language (new constructs for aspects).
- Produces standard Java bytecode.
- Weaves into class files.
AspectJ example: pre-condition

/** Java code */
class Test {
    public int val ;
    /* ... */
}

/* AspectJ code */
aspect Precondition {
    public static final int MAX_VALUE = 2000 ;
    before(int newval): set (int Test.val) && args (newval) {
        if (newval > MAX_VALUE)
            throw new RuntimeException("Invalid value");
    }
}
**AspectJ example: persistence**

```java
aspect DatabaseAspect {
    pointcut transactionalMethods ():
        execution (/* pattern for transactional methods */) ;

    before (): transactionalMethods () {
        initialiseDatabase() ;
    }

    after() returning: transactionalMethods() {
        commitTransaction() ;
    }

    after() throwing: transactionalMethods() {
        rollbackTransaction() ;
    }
}

(Soares et. al. Implementing distribution and persistence aspects with AspectJ.)
```
**AspectJ – Pointcuts**

- Pointcuts define where to intervene
- Expressed from primitive pointcuts:
  - `call/execution(<Method signature>)`
  - `set/get(<Field signature>)`
  - `cflow(<Pointcut>)`
  - `args, target, this`

- `pointcut setvalue(int val):
call(public void set*(int)) && args(val);`
Application: Design patterns
Observer pattern (I)

- In Java, explicit “addObserver” and “notify” embedded the observed class.
- “Observable” feature tangled with the main purpose of the class.
- ```java
public void process() {
    /* Do something */
    notifyObservers();
}
```
Observer pattern (II)

- In AspectJ, possible to decouple the “observable” from the main purpose of the class.

  ```java
  after(): execution(* *.process(..)) {
      /* notify or perform an action */
  }
  ```

- The “observer” and “observable” can be encapsulated into a single aspect.
Conclusions


Benefits and pitfalls

- **Benefits:** clearer decomposition of the roles (more reusability)

- **Pitfalls:**
  - Learning curve to comprehend the concepts (eased by Java environment)
  - Need for tools to understand the overall behaviour of the application (Eclipse AJDT)
    - (available for other IDEs as well)
References (I)

- AOSD website: http://www.aosd.net/
- AspectJ: http://www.eclipse.org/aspectj/
- “Eclipse AspectJ” by A. Colyer et al., ISBN: 0321245873
- [Fil05] “Aspect-Oriented Software Development” by R. Filman et al., ISBN: 0321219767
References (II)

- “Towards a Catalog of Aspect-Oriented Refactorings”, by M. Monteiro et al., AOSD 2005.