

AOP examples

Paolo Falcarin



SoftEng
<http://softeng.polito.it>

Corso 3° livello

2007

Aspects

- Unit that combines pointcuts and advices
- Can contain methods and fields
- Can extend classes or implement interfaces
- Cannot create an ‘aspect object’ using *new*
- Aspects and pointcuts can be abstract

Abstract Aspects

- A generic aspect requiring authentication through a singleton Authenticator:

```
// AbstractAuthenticationAspect.java
public abstract aspect AbstractAuthenticationAspect {
    public abstract pointcut opsNeedingAuthentication();
    before() : opsNeedingAuthentication() {
        // Perform authentication. If not authenticated,
        // let the thrown exception propagate.
        Authenticator.authenticate();
    }
}
```

Abstract Aspects II

- A concrete aspect for a database app:

```
// DatabaseAuthenticationAspect.java
public aspect DatabaseAuthenticationAspect
    extends AbstractAuthenticationAspect {

    public pointcut
    opsNeedingAuthentication():
        call(* DatabaseServer.connect());
}
```

Logging

```
aspect Logger {
    pointcut logPoint(): call(* *.*(..));
    before() : logPoint && !within(Logger+) {
        System.out.println(thisJoinPoint.getTarget() + "," +
            thisJoinPoint.getThis() + "," +
            thisJoinPoint.getSignature());
    }
}
```

Design by contract

- Each method must fulfill pre-conditions, post-conditions and invariants.
- When accessing a DB, the typical precondition is:
 - ◆ input data must not be null
- AOP solution:
 - ◆ Insert code at the beginning of each method that ensures holding of pre-conditions, before a method begins executing.

Null Checker with AspectJ

```

public aspect NullChecker {
    pointcut arguments():execution( * db.*.* (..));
    before(): arguments() {
        Object args[ ] = thisJoinPoint . getArgs();
        for (int j=0; j<args.length; j++){
            if (args[ j ]==null)
                throw new IllegalArgumentException();
        }
    }
}

```

Diagram annotations for the AspectJ code:

- Return value: points to the return value of the `execution` function.
- parameters: points to the parameters of the `execution` function.
- Pointcut definition: points to the `pointcut arguments():` line.
- Pointcut used in advice: points to the `arguments()` parameter in the `before()` method.
- package: points to the `db` package in the pointcut.
- class: points to the `*` representing a class in the pointcut.
- method: points to the `*` representing a method in the pointcut.

SoftEng
http://softeng.polito.it

Paolo Falcarin

Example: Design by contract

- Each method must fulfill pre-conditions, post-conditions and invariants.
- Method parameters' values must satisfy pre-conditions
- Typically, graphical element, like **Point**, coordinates must be included in window boundaries:
- AOP solution:
 - ♦ Insert code at the beginning of each method that ensures holding of pre-conditions, before a method begins executing.

SoftEng
http://softeng.polito.it

Paolo Falcarin

Pre- and Post- Conditions

aspect PointBoundsChecking {

pointcut setX(int x): (**call**(void Point.setXY(int,int))
&&args(x, *)) || (**call**(void Point.setX(int)) && args(x));
pointcut setY(int y): (**call**(void Point.setXY(int,int)) &&
args(*, y)) || (**call**(void Point.setY(int)) && args(y));

before(int x): setX(x) {
 if (x < MIN_X || x > MAX_X) throw new
 IllegalArgumentException("x is out of bounds."); }

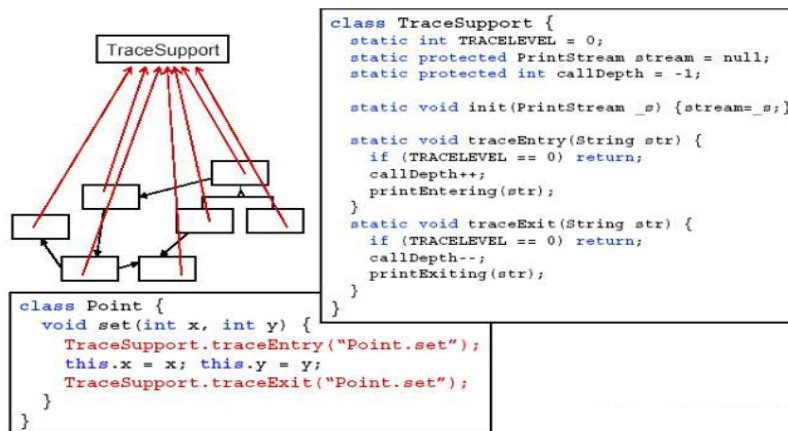
before(int y): setY(y) {
 if (y < MIN_Y || y > MAX_Y) throw new
 IllegalArgumentException("y is out of bounds."); }

}

SoftEng
http://softeng.polito.it

Paolo Falcarin

Tracer without AspectJ

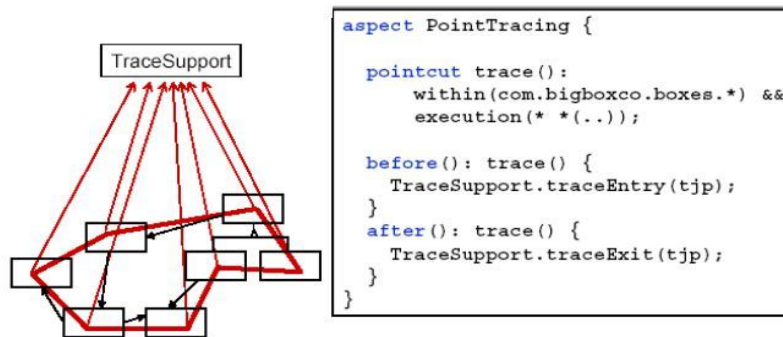


http://softeng.polito.it

Paolo Falcarin

Tracing as an aspect

- All modules of the system use the trace facility in a consistent way:
 - ◆ entering the methods and exiting the methods



<http://softeng.polito.it>

Paolo Falcarin

Benefits of aspects in the design

- objects are no longer responsible for using the trace facility
 - ◆ trace aspect encapsulates that responsibility, for appropriate objects
- if the Trace interface changes, that change is shielded from the objects
 - ◆ only the trace aspect is affected
- removing tracing from the design is trivial
 - ◆ just remove the trace aspect

SoftEng
<http://softeng.polito.it>

Paolo Falcarin

Benefits of aspects in the code

- object code contains no calls to trace functions
 - ◆ trace aspect code encapsulates those calls, for appropriate objects
- if the Trace interface changes, there is no need to modify the object classes
 - ◆ only the trace aspect class needs to be modified
- removing tracing from the application is trivial
 - ◆ compile without the trace aspect class

Example: mock objects with AspectJ

Mock objects (i.e. dummy server stubs) aid unit testing

But some work is needed:


- They must have common interface for mock and real
- Might need configuration service to use mock or real
- This aspect trap calls to Player with no need for common interface or special configuration:

```
public aspect MockPlayer {  
    private Square square = new Square("go");  
    String around(): call( String Player.getName() ) {  
        return "Paolo";  
    }  
    String around(): call( String Player.getLocation() ) {  
        return square;  
    }  
}
```

around() advice replaces method implementation

Connections Pooling (1)

```
import java.sql.*; import java.util.*;
public aspect Pooling {
private static Stack pool = new Stack();
pointcut poolGet():
call(static Connection DriverManager.getConnection(..));
Connection around() throws SQLException: poolGet() {
synchronized(pool) {
if(pool.empty()) {
return proceed();
}
return (Connection) pool.pop();
}
}
}
```



Paolo Falcarin

Connections Pooling (2)

```
pointcut poolPut(): call(void Connection.close());
void around(): poolPut() {
Connection conn =(Connection) thisJoinPoint.getTarget();
pool.push(conn);
}}
```

- This example contains a stack of connections
- Override *getConnection* and *close* methods.
- When the system is finished the connection (instead of being disconnected) will be placed in the pool so it can be reused.



Paolo Falcarin

Asynchronous call

```
public aspect Async {
    private ThreadPool pool = ...;
    pointcut methods(): call( Service1.*(..));
    Object around() : methods() {
        Runnable r = new Runnable() { public void run() {
            try {
                // invoke the real method within a thread
                proceed();
            } catch (Throwable e) {
                throw e;
            }
        }};
        r.start();
    }
}
```

SoftEng
http://softeng.polito.it

Paolo Falcarin

Profiling

- It's easy to ask very specific questions, and quickly modify them, all outside the real code

```
aspect SetsInRotateCounting {
    int rotateCount = 0;
    int setCount = 0;
    before(): call(void Line.rotate(double)) {
        rotateCount++; }
    before(): call(void Point.set*(int) &&
        cflow(call(void Line.rotate(double))) {
        setCount++; }
}
```

SoftEng
http://softeng.polito.it

Paolo Falcarin

Contract Enforcement

- Useful to check assertions, use Design by Contract,
- Ex: only certain factory methods can put objects in a central Registry

```
aspect RegistrationProtection {
    pointcut register():
        call(void Registry.register(Element));
    pointcut canRegister():
        withincode(static * Element.make*(..));
    before(): register() && !canRegister() {
        throw new IllegalAccessException("Illegal call "
+thisJoinPoint); }
}
```

SoftEng
http://softeng.polito.it

Paolo Falcarin

Pointcuts cflow and cflowbelow

- **cflow**(call(void Figure.move())
 - ♦ Any join point in the control flow of each call to *void Figure.move()*
 - ♦ This includes the call itself.
- **cflowbelow**(call(void Figure.move())
 - ♦ Any join point **below** in the control flow of each call to *void Figure.move()*
 - ♦ This does not include the call.

SoftEng
http://softeng.polito.it

Paolo Falcarin

Example: all joinpoints in a call–stack range

```
public aspect PrintJoinPoints {
    pointcut begin(): execution( * *.main(..));
    pointcut end(): execution( void Foo.bar(..));

    before():
    cflow( begin() )
    && !cflowbelow(end() )
    && !within( PrintJoinPoints ) {
        System.out.println(thisJoinPoint);
    }
}
```



Paolo Falcarin

Introductions

- Modifying the static form of a class
- Add fields to an existing class
 - ◆ private boolean Server.disabled = false;
 - ◆ public String Foo.name;
- Add methods to an existing class
 - ◆ public int Point.getX() { return x; }
 - ◆ public String (Point || Line).getName() {return name;}
- Add Constructors
 - ◆ public Point.new(int x, int y) {



```
    this.x = x; this.y = y; }
```

Paolo Falcarin

Introductions II

- Extend an existing class with another
 - ◆ **declare parents**: Point **extends** GeometricObject;
- Implement an interface with an existing class
 - ◆ **declare parents**: Point **implements** Comparable;
- “Soften” Exception
 - ◆ Convert checked exceptions to unchecked ones
 - ◆ Wraps exceptions in `org.aspectj.lang.SoftException`
 - ◆ **declare soft**: CloneNotSupportedException:
 `execution(Object clone());`

Introductions III

- Add a compile-time warning or error
- Issued if there is a chance that code will reach a given pointcut
- Warning / error string can be defined
- **declare warning**: *Pointcut: String*;
- **declare error**: *Pointcut: String*;

Code segregation: DB example

```
public aspect CodeSegregation {
    pointcut dbCode():
        call(ConnectionDriverManager.getConnection(..));
    pointcut badDbCode(): dbCode() && !within(db.*);
    pointcut reallyBadDbCode(): badDbCode() &&
        !within(security.*) && !within(servlets.*);
    declare warning: badDbCode() :
        "Database code outside 'db' package.";
    declare error : reallyBadDbCode():
        "Database code here is not permitted.";
}
```

SoftEng
http://softeng.polito.it

Paolo Falcarin

Design Pattern through AOP

- E.g. **Factory method** pattern: for each “*constructor call*” joinpoint defined in a class in package *pack*, not defined in a *Factory** class => write compile error on output

```
public aspect FactoryEnforcer {
    declare error:
        within (pack..*)
        && call (pack..Pippo.new(..))
        && !withincode(* Factory*.*(..)) :
```

“Aspect enforcing ERROR: new operator used outside of a factory. getInstance method”

} **SoftEng**
http://softeng.polito.it

Declare a new kind
of compile error

Output Message
of compiler

Paolo Falcarin

Flexible Access Control

- Control method access beyond *private*, *protected* and *public* declarations
- Violations must be found at compile time
- For example, class Product can only be initialized and configured by specific classes

Flexible Access Control II

- Use **declare error** inside a nested aspect

```
aspect FlagAccessViolation {  
    pointcut factoryAccessViolation():  
        call(Product.new(..)) && !within(ProductFactory+);  
    pointcut configuratorAccessViolation():  
        call(* Product.configure(..)) &&  
            !within(ProductConfigurator+);  
    declare error: factoryAccessViolation() ||  
                   configuratorAccessViolation()  
                   : "Access control violation";  
} // end of aspect
```

Subject & Observer (1)

The following aspect is an ‘automatic’ observer for Screen objects observing Points

aspect PointObserving {

```
private Vector Point.observers = new Vector();
public static void addObserver(Point p, Screen s) {
    p.observers.add(s); }
public static void removeObserver(Point p, Screen s) {
    p.observers.remove(s); }
static void updateObserver(Point p, Screen s) {
    s.display(p); }
```

SoftEng
http://softeng.polito.it

Paolo Falcarin

Subject & Observer (2)

pointcut changes(Point p):
 target(p) && **call**(void Point.set*(int));

after(Point p): changes(p) {
 Iterator iter = p.observers.iterator();
 while (iter.hasNext()) {
 updateObserver(p, (Screen)iter.next()); }
}

- This can be easily generalized
- Same pattern is useful for logging, profiling, ...

SoftEng
http://softeng.polito.it

Paolo Falcarin

Making a Class Cloneable

- Given a standard *Point* class, with private fields x, y we can make it cloneable:

```
aspect CloneablePoint {  
    declare parents: Point implements Cloneable;  
    declare soft: CloneNotSupportedException:  
        execution(Object clone());  
    Object Point.clone() { return super.clone(); }  
}
```
- The same technique can be used for any mixin that has a standard implementation

More Aspect Features

- You can declare an aspect as dominant, in case several aspects affect the same join point
 - ◆ **declare precedence:** *Aspect List* ;
- Aspects don't have to be singletons
 - ◆ Useful in case they contain fields
 - ◆ *perthis, pertarget, perflow, perflowbelow*
- Aspects can be allowed to use private fields of aspected classes
 - ◆ The *privileged* keyword

Functional Guidelines

- Code of aspected classes doesn't change
- Multiple aspects can co-exist
- Same pattern is useful for many other cases
 - ◆ Security, Authentication
 - ◆ Resource Pooling, Caching, Copy on write, ...
 - ◆ Creation by Factory, Lazy Creation, ...
 - ◆ Multi-Thread Synchronization
 - ◆ Transaction Definition
 - ◆ Monitoring System Notification
 - ◆ Standard Exception Handling

Categories of Aspects

- . Development
 - . Logger, Profiler, Tracer, Smart Debugger, Testing, Design Patterns
- . Production
 - . Authentication & Security, Exceptions, Design by Contract, Middleware, Synchronization
- . Runtime and optional
 - . Pooling, ConnectionChecking, Caching, Performance improvement