Aspect-Oriented Programming and AspectJ

CUGS
Mikhail CHALABINE
mikch@ida.liu.se

Outline
Introduction to AOP
AspectJ

Look for answers

• What are the new concepts introduced?
  • What is a crosscutting concern?
  • What is an aspect?
  • What is dynamic aspect weaving?
  • What is static aspect weaving?
  • What is a join point?
  • What is a dynamic join point model?
  • What is a static join point model?

• What are the problems, benefits?

A Big Picture

• Reification
• Reflection
  • Introspection
  • Introcession

Java
Compost, AOP, etc.
Object Oriented Programming

Objects model the real world
Data and operations combined
Encapsulation
Objects are self contained

Separation of concerns?

Example (1)

```java
class Account {
    private int balance = 0;
    public void deposit(int amount) {
        balance = balance + amount;
    }
    public void withdraw(int amount) {
        balance = balance - amount;
    }
}
```

Example (2)

```java
class Logger {
    private OutputStream stream;
    Logger() {
        // Create stream
    }
    void log(String message) {
        // Write message to stream
    }
}
```

Example (3)

```java
class Account {
    private int balance = 0;
    Logger logger = new Logger();
    public void deposit(int amount) {
        balance = balance + amount;
        logger.log("deposit amount: " + amount);
    }
    public void withdraw(int amount) {
        balance = balance - amount;
        logger.log("withdraw amount: " + amount);
    }
}
```
What is Crosscutting

Code in objects (components, programs) not directly related to the core functionality
- User authentication
- Persistence
- Timing

Mixing of concerns leads to
- Code scattering
- Code tangling

Problems (intermixed concerns)

- Correctness
- Understandability
- Testability
- Maintenance
  - Find code
  - Change it consistently
  - No help from OO tools
- Reuse

XML parsing

XML parsing in org.apache.tomcat

URL pattern matching

URL pattern matching in org.apache.tomcat
Logging

logging in org.apache.tomcat

Aspect Oriented Programming

Aspect ≈ Implementation of a crosscutting concern
Components and component language
Aspects and aspect language
Does not replace OOP
Code does not have to be OO based

BIG PICTURE

Back to the example

```java
class Account {
    private int balance = 0;

    public void deposit(int amount) {
        balance = balance + amount;
    }

    public void withdraw(int amount) {
        balance = balance - amount;
    }
}
```
Weave on demand

Automatically transform given (program) code

```java
aspect Logging {
    Logger logger = new Logger();
    WHENEVER ANY METHOD IS CALLED () {
        logger.log("Method is called");
    }
}
```

Example big picture

![Diagram showing Account, Logger, and System with logging connected by 'WEAVER' and 'Composite' nodes.]

Weaving Time

- Preprocessor
- Compile time
- Link time
- Load time
- Run time

New concepts

- Weaving
- Join point
- Pointcut
- Advice
- Aspect
**Join Point**

**Static join point model**
A location in (a component) code where a concern crosscuts

**Dynamic join point model (AspectJ)**
A well-defined point in the program flow

Examples:
- Method / class declaration
- A call to a method

```
public void Account.deposit(int)
```

**Pointcut**

A **pointcut** picks out certain join points

- Specifies join points
- Exposes parameters at join points
- Is a predicate that matches join points

Example

The `balanceAltered()` pointcut picks out each join point that is a call to either the `deposit()` or the `withdraw()` method of an Account class

```
pointcut balanceAltered() :
   call(public void Account.deposit(int)) ||
        call(public void Account.withdraw(int));
```

**Pointcut (further examples)**

- `call(void SomeClass.make*(..))`
  - picks out each join point that’s a call to a void method defined on `SomeClass` whose name begins with “make” regardless of the method’s parameters
- `call(public * SomeClass.* (..))`
  - picks out each call to `SomeClass` public methods
- `cflow(somePointcut)`
  - picks out each pointcut that occurs in the dynamic context of the join points picked out by `somePointcut` pointcuts in the control flow, e.g., in a chain of method calls

**Advice**

Code executed at a pointcut (join point reached, joint point is matched)

```
before(int i) : balanceAltered(i) {
    System.out.println("The balance changed");
}
```
**Aspect**

The unit of modularity for a crosscutting concern
Implements join points, pointcuts, advice

```java
public aspect LoggingAspect {
    pointcut balanceAltered(int i) :
        call(public void Account.deposit(int)) ||
        call(public void Account.withdraw(int));

    before(int i) : balanceAltered(i) {
        System.out.println("The balance changed");
    }
}
```

**So far we have**

Agreed that tangled, scattered code that appears as a result of mixing different crosscutting concerns in (OO) programs is a problem
Sketched a feasible solution - AOP

Introduced
- Join points
- Pointcuts
- Advice
- Aspects
- Weaving

Tools?

**AspectJ**

Xerox Palo Alto Research Center
Gregor Kiczales, 1997
Goal: Make AOP available to developers
  - Open Source
  - Tool integration Eclipse
Java with aspect support
Current focus: industry acceptance

**Join Points**

- Method call execution
- Constructor call execution
- Field get
- Field set
- Exception handler execution
- Class/object initialization
Patterns as Regular expressions

- Match any type: *
- Match 0 or more characters: *
- Match 0 or more parameters: (...) 
- All subclasses: Person+
- Call: call(private void Person.set*(*)
- Call: call(* * *.*(*))
- Call: call(* * *.*(..))

Logical Operators

- Match all constructor-based instantiations of subclasses of the Person class
  
  call((Person+ && ! Person).new(..))

Pointcut Example

- Match all attempts to retrieve the balance variable of the Account class

  pointcut balanceAccess() :
  get(private int Account.balance);

Exposing Context in Pointcuts (1)

- Matching with parameters
  AspectJ gives code access to some part of the context of the join point (parts of the matched pattern)
  Two ways
  - Methods
  - Designators
Exposing Context in Pointcuts (2)

thisJoinPoint class and its methods

Designators

State-based: this, target, args
Control Flow-based: cflow, cflowbelow
Class-initialization: staticinitialization
Program Text-based: withincode, within
Dynamic Property-based: If, adviceexecution

Exposing Context in Pointcuts (3)

Methods

getThis()
getTarget()
getArgs()
getSignature()
getSourceLocation()
getKind()
toString()
toShortString()
toLongString()

Exposing Context in Pointcuts (4)

Example

```java
public class DVD extends Product {
    private String title;
    ...
}

public aspect OutputType {
    pointcut callToDVDConstructor(): call((DVD).new(..));
    before(): callToDVDConstructor() { 
        SourceLocation sl = thisJoinPoint.getSourceLocation(); Class theClass = (Class) sl.getWithinType();
        System.out.println(theClass.toString());
    }
}

Output: class DVD
```

Designators (1)

Execution - Matches execution of a method or constructor
Call - Matches calls to a method
Initialization - Matches execution of the first constructor
Handler - Matches exceptions
Get - Matches the reference to a class attribute
Set - Matches the assignment to a class attribute
Designators (2)

This
- Returns the object associated with a particular join point or limits the scope of a join point by using a class type

Target
- Returns the target object of a join point or limits the scope of join point

Args
- Exposes the arguments to a join point or limits the scope of the pointcut

Designators (3)

Cflow
- Returns join points in the execution flow of another join point

Cflowbelow
- Returns join points in the execution flow of another join point but including the current join point

Staticinitialization
- Matches the execution of a class's static initialization

Designators (4)

Withincode
- Matches within a method or a constructor

Within
- Matches within a specific type (class)

If
- Allows a dynamic condition to be part of a pointcut

Adviceexecution
- Matches on advice join points

Preinitialization
- Matches pre-initialization join points

One More Exposing Context Example

```java
pointcut setXY(FigureElement fe, int x, int y):
    call(void FigureElement.setXY(int, int))
    && target(fe) && args(x, y);

...}

after(FigureElement fe, int x, int y) returning: setXY(fe, x, y)
{
    System.out.println(fe +
           " moved to (" + x + ", " + y + ").");
}
```
Exposing Context Comment

Prefer designators over method calls
Higher cost of reflection associated with get*

```java
pointcut setXY():
    call(void FigureElement.setXY(int, int));
after() returning: setXY() {
    FigureElement fe = thisJoiningPoint.getThis();
    ... System.out.println(fe + " moved to (" + x + ", " + y + ").");
}
```

Advice

Before
After
Unqualified
After returning
After throwing
Around

BEFORE Advice Example

```java
pointcut withdrawal() :
    call(public void Account.withdraw(int));
...
before() : withdrawal() {
    // advice code here
}
```

AFTER Advice Example

```java
pointcut withdrawal() :
    call(public void Account.withdraw(int));
...
after() : withdrawal() {
    // advice code here
}
```
AFTER RETURNING Advice Example

```java
pointcut withdrawal() :
    call(public void Account.withdraw(int));
...

after() returning : withdrawal() {
    // advice code here
}
```

AFTER THROWING Advice Example

```java
pointcut withdrawal() :
    call(public void Account.withdraw(int));
...

after() throwing(Exception e) : withdrawal() {
    // advice code here
}
```

AROUND Advice Example

```java
pointcut withdrawal() :
    call(public void Account.withdraw(int));
...

around() : withdrawal() {
    // do something
    proceed();
    // do something
}
```

Inter-type Declarations

So far we assumed dynamic join point model

Inter-type Declarations assume static program structure modification
- Static joint point model
- Compile-time weaving
Inter-type Declarations

Add members
   methods
   constructors
   fields
Add concrete implementations to interfaces
Declare that types extend new types
Declare that types implement new interfaces

Other AOP languages

AspectWerkz
JAC
JBoss-AOP
Aspect#
LOOM.NET
AspectR
AspectS
AspectC
AspectC++
Pythius

Possible applications

Resource pooling connections
Caching
Authentication
Design by contract
Wait cursor for slow operations
Inversion of control
Runtime evolution