Aspect Oriented Programming

Neill Rolando Giraldo Corredor\textsuperscript{1}
Iván Darío Vanegas Pérez\textsuperscript{1}

1. Facultad de Ingeniería, Departamento de Sistemas e Industrial
Content

● Introduction
  ○ Main Goal
  ○ Emerging Programming Problems
  ○ Advanced Separation of Concerns
  ○ Programming Paradigms

● History & Previous Work
  ○ Brief Timeline
  ○ Reflection
  ○ Metaobjects Protocol
  ○ Subject Oriented Programming
  ○ Composition Filters
  ○ Adaptive Programming
Content

- Concepts
  - Aspect
  - JoinPoint
  - Advice
  - CutPoint
  - Target
  - Proxy
  - Weaving
- Development Example
- Advantages & Disadvantages
- Criticism
- Conclusions
- Review
- Bibliography
INTRODUCTION
The main goal of this paradigm is to separate into well-defined modules the core functionalities and logic data details of the whole system and its components from those of common use across them.
public class SomeBusinessClass extends OtherBusinessClass {
    // Core data members
    // Other data members: Log stream, data-consistency flag
    // Override methods in the base class
    public void performSomeOperation(OperationInformation info) {
        // Ensure authentication
        // Ensure info satisfies contracts
        // Lock the object to ensure data-consistency in case other
        // threads access it
        // Ensure the cache is up to date
        // Log the start of operation
        // ==== Perform the core operation ====
        // Log the completion of operation
        // Unlock the object

        // More operations similar to above
        public void save(PersistenceStorage ps) {
            // ...
        }

        public void load(PersistenceStorage ps) {
            // ...
        }
    }
}
Programming Problematic

- **Scattered Code**
  - Functionalities that appear on more than one class, method or entity
  - Security-operations

- **Tangled Code**
  - Different mixes of code across basic functionality-line that imply a hard-to-follow execution flow
  - Transactions, logging

*Concern := set of information (from data or processes) that has an effect on the code of a computer application*
Advanced Separation of Concerns (ASoC)

- **Concerns**
  - Main Functionality
  - Common Functionality - Crosscutting Concerns

- **Advantages**
  - Clarity
  - Adaptability
  - Maintainability
  - Scalability
  - Reusability
Programming Paradigms

- 1st: Procedural
- 2nd and 3rd: Functional Decomposition;
- 4th: POO

Fig 3. http://images.slideplayer.es/12/3583443/slides/slide_4.jpg

Fig 4 & 5. http://images.slideplayer.es/12/3583443/slides/slide_4.jpg
HISTORY & PREVIOUS WORK
   Introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.

1993: Introduction of Subject Oriented Programming

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales introduced AOP concepts

199X: Introduction of Subject Oriented Programming

1993: Xerox PARC designed AspectJ

2001: AspectC++

2001: IBM designed HyperJ

2001: IBM designed AspectJ

2006: phpAspect
Timeline

1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns.”


1982: Introduction of Reflection in Procedural PL.

1994: Introduction of Composition Filter Object Model

1993: Introduction of Subject Oriented Programming

1997: Gregor Kiczales Introduced of AOP concepts

199X: Introduction of Adaptive Programming

2001: Xerox PARC designed AspectJ

2001: IBM designed HyperJ

2001: AspectC++ Aspect (Perl)

2006: phpAspect
1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.

1993: Introduction of Subject Oriented Programming

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales Introduced of AOP concepts

199X: Introduction of Subject Oriented Programming

1997: Gregor Kiczales Introduced of AOP concepts

2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl)

2001: IBM designed HyperJ

2006: phpAspect
Meta-Object Protocol

A ‘MOP’ Provides the vocabulary (protocol) to access and manipulate the structure and behaviour of systems of objects

- Create or delete a new class
- Create a new property or method
- Cause a class to inherit from a different class ("change the class structure")
- Generate or change the code defining the methods of a class
How many levels of Depth levels can recursively declare a MetaObject?

- None, a metaobject cannot declare anything recursively
- Several
- Depends on meta-details nature specification
1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.

1993: Introduction of Subject Oriented Programming

1993: Gregor Kiczales Introduced of AOP concepts

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales Introduced of AOP concepts

199X: Introduction of Adaptive Programming

2001: Xerox PARC designed AspectJ

2001: AspectC++ Aspect (Perl)

2001: IBM designed HyperJ

2001: IBM designed HyperJ

2006: phpAspect
Reflection

“Reflection is the ability of a computer program to examine, introspect and modify its own structure or behavior at runtime”[1]

In OO programming languages reflection allows:

● Inspection:
  ○ Classes
  ○ Interfaces
  ○ Fields
  ○ Methods

● Instantiation of Objects
● Invocation of Methods

PROGRAMMING EXAMPLES
Is it possible to add dynamic code after compile time only using reflection in Java?

- Yes
- No
Timeline

- Introduced the concept "Separation of Concerns"


1982: Introduction of Reflection in Procedural PL.

1993: Introduction of Subject Oriented Programming

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales
- Introduced of AOP concepts

199X: Introduction of Adaptive Programming

2001: IBM designed HyperJ

2001: Xerox PARC designed AspectJ

2001: AspectC++
- Aspect (Perl)

2006: phpAspect
Subject Oriented Programming

“It is an object-oriented software paradigm in which the state and behavior of objects are not seen as plain objects, but the perceptions of themselves”[1]

Philosophical analogy of Plato over ideal & real world applied to software.

- An object exists because is perceived by another object => Subjects.

1970: Edsger W. Dijkstra introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.


1993: Introduction of Subject Oriented Programming

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales introduced AOP concepts

199X: Introduction of Adaptive Programming

2001: Xerox PARC designed AspectJ

2001: IBM designed HyperJ

2006: phpAspect
Composition Filters

“Composition filters changes the behavior of an object through the manipulation of incoming and outgoing messages.”

- Design of a Composition Filter
  - Kernel or Implementation Part
  - Outer layer or Interface Part
1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.

1993: Introduction of Subject Oriented Programming

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales introduced AOP concepts

199X: Introduction of Adaptive Programming

2001: Xerox PARC designed AspectJ

2001: IBM designed HyperJ

2001: AspectC++

2006: Aspect (Perl)

2006: phpAspect
Adaptive Programming

Finite state machine graph flow for efficiently move through sets of paths.

“communicate only to nearby neighbors”

- Shy system concerns
- Loose Coupling
- Previous to POO
- DEMETER LAW

http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/Pages/historia.html
1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.

1993: Introduction of Subject Oriented Programming

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales Introduced of AOP concepts

199X: Introduction of Adaptive Programming

2001: AspectC++

2001: IBM designed HyperJ

2001: Xerox PARC designed AspectJ

2006: phpAspect
Timeline

- **1970:** Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”
- **1980:** SmallTalk-80 introducing Meta Object Protocol.
- **1982:** Introduction of Reflection in Procedural PL.
- **1993:** Introduction of Subject Oriented Programming
- **1994:** Introduction of Composition Filter Object Model
- **1997:** Gregor Kiczales introduced AOP concepts
- **199X:** Introduction of Adaptive Programming
- **2001:** Xerox PARC designed AspectJ
- **2001:** IBM designed HyperJ
- **2001:** IBM designed AspectC++ Aspect (Perl)
- **2006:** phpAspect
Is it true that AP evolves from AOP?

- Yes
- No
1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.

1993: Introduction of Subject Oriented Programming

1994: Introduction of Composition Filter Object Model

1997: Gregor Kiczales Introduced of AOP concepts

199X: Introduction of Adaptive Programming

2001: Xerox PARC designed AspectJ

2001: IBM designed HyperJ

2001: ASpectC++ ASpect (Perl)

2006: phpAspect
Timeline

1970: Edsger W. Dijkstra. Introduced the concept “Separation of Concerns”


1982: Introduction of Reflection in Procedural PL.

1983: Introduction of Composition Filter Object Model

1993: Introduction of Subject Oriented Programming

1997: Gregor Kiczales Introduced of AOP concepts

199X: Introduction of Adaptive Programming

2001: Xerox PARC designed AspectJ

2001: IBM designed HyperJ

2001: AspectC++ Aspect (Perl)

2006: phpAspect
Who was the founder of the Aspect Oriented Programming?

- Edsger W. Dijkstra
- IBM
- Gregor Kiczales
Main Concepts
Aspect

What is an Aspect?

- A component that can not be fully encapsulated into a generalized procedure that can either be:
  - Method
  - Object
  - API.

- Tend to be non-functional decomposition of the system, that usually affects performance or semantic of the System.
Aspect

What is an Aspect?

- **A modular unit** that disseminate through different functional unit of a system (Crosscut).

- Are the implementation of what is known as CrossCut Concerns.

- Identifying Aspects and applying Aspect Oriented Programming techniques an adequate Concern Separation can be accomplished easier.
Join Point

what is a Join Point?

- Possible places of actual business logic execution flow where advices can be executed
- Defined on object component Class
cross-cutting concepts

System’s basic functionality

Fig 5. http://2.bp.blogspot.com/-KPr3lQ2BMgE/TatFHX8DzbI/AAAAAAAABIM/wFV0h4M1gbY/s1600/poa3.JPG
Advices

What is an Advice?

- Actions taken at a given Join Point.
- Can be seen as methods that are executed when a certain Join Point with matched Cut Point is reached in the application to add extra code.
Fig 5. http://2.bp.blogspot.com/-KPr3lQ2BMgE/TatFHX8DzbI/AAAAAAAABIM/wFV0h4M1gbY/s1600/poa3.JPG
Advice Types

● Before Advice
  ○ Executed before the Join Point method.

● After Advice
  ○ Executed when the Join Point method finished whether normally or by an exception.

● Return Advice
  ○ Executed when the Join Point method finished normally.

● Throwing Advice
  ○ Executed when the Join Point method finished by an exception.

● Around Advice
  ○ Can be seen as the All-In-Advice. Can manage Join Point methods call and advices surrounding them
What type of advice would you use to apply authentication?

- Before Advice
- After Advice
- Return Advice
- Throwing Advice
- Around Advice
Cut Point

- Instruction that matches a `JoinPoint` by regular expressions
- Defined on `Aspect Class`

```java
class ExampleBusinessClass {
  public Object doYourBusiness() { return new Object(); }
}

@Aspect
class SomeAspect {

  @Pointcut("execution(* com.amanu.example.ExampleBusinessClass.doYourBusiness())")
  public void somePointCut() {
    //Empty body suffices
  }

  @After("somePointCut()")
  public void afterSomePointCut() {
    //Do what you want to do before the joint point is executed
  }

  @Before("execution(* *(*)") public void beforeSomePointCut() {
    //Do what you want to do before the joint point is executed
  }
}
```
<table>
<thead>
<tr>
<th>Pointcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>call(public void MyClass.myMethod(String))</code></td>
<td>Call to <code>myMethod()</code> in <code>MyClass</code> taking a <code>String</code> argument, returning <code>void</code>, and with <code>public</code> access</td>
</tr>
<tr>
<td><code>call(void MyClass.myMethod(..))</code></td>
<td>Call to <code>myMethod()</code> in <code>MyClass</code> taking any arguments, with <code>void</code> return type, and any access modifiers</td>
</tr>
<tr>
<td><code>call(*) MyClass.myMethod(..)</code></td>
<td>Call to <code>myMethod()</code> in <code>MyClass</code> taking any arguments returning any type</td>
</tr>
<tr>
<td><code>call(*) MyClass.myMethod*(..)</code></td>
<td>Call to any method with name starting in &quot;myMethod&quot; in <code>MyClass</code></td>
</tr>
<tr>
<td><code>call(*) MyClass.myMethod*(String,..)</code></td>
<td>Call to any method with name starting in &quot;myMethod&quot; in <code>MyClass</code> and the first argument is of <code>String</code> type</td>
</tr>
<tr>
<td><code>call(*) *.*.myMethod(..)</code></td>
<td>Call to <code>myMethod()</code> in any class in default package</td>
</tr>
<tr>
<td><code>call(MyClass.new())</code></td>
<td>Call to any <code>MyClass</code>'s constructor taking no arguments</td>
</tr>
<tr>
<td><code>call(MyClass.new(..))</code></td>
<td>Call to any <code>MyClass</code>'s constructor with any arguments</td>
</tr>
<tr>
<td><code>call(MyClass+..new(..))</code></td>
<td>Call to any <code>MyClass</code> or its subclass's constructor. (Subclass indicated by use of <code>+</code> wildcard)</td>
</tr>
<tr>
<td><code>call(public * com.mycompany..*.*(..))</code></td>
<td>All public methods in all classes in any package with <code>com.mycompany</code> the root package</td>
</tr>
</tbody>
</table>
According to the Join Points model, the behaviour of system methods can be altered by advices at:

- Precompile time only
- Mostly at runtime
Introduction

What is an Introduction?

- Allows adding new attributes or methods to existing classes.

- Python Example.
PROGRAMMING EXAMPLES
from Logging import Logger

class User:

    def __init__(self, name, password):
        self.name = name
        self.password = password

@Logger.logMethod
    def sayHi(self):
        return "Hi"

@Logger.logMethod
    def sayGoodBye(self):
        return "Goodbye"
import functools

class Logger:

    @staticmethod
    def logMethod(func):
        @functools.wraps(func)
        def decorator(self, *args, **kwargs):
            func(self, *args, **kwargs)
            attr = dir(self)
            if "logger" not in attr:
                self.logger = []
            if func.func_name == "sayHi":
                self.logger.append(self.name + " said Hi")
            if func.func_name == "sayGoodBye":
                self.logger.append(self.name + " said GoodBye")

        return decorator
from User import User
from Logging import Logger

logger = Logger()

neill = User("Neill", 1234)
ivan = User("Ivan", 1234)
pancho = User("Pancho", 1234)
ignacio = User("Ignacio", 1234)
sara_abril = User("Sara Abril", 1234)

neill.sayHi()
ivan.sayHi()
pancho.sayHi()
ignacio.sayHi()

neill.sayGoodBye()
sara_abril.sayHi()
pancho.sayGoodBye()
ivan.sayHi()
sara_abril.sayGoodBye()
pancho.sayHi()
ivan.sayGoodBye()

logger.get_logs(pancho)
What crosscut-concerns were present at the previous example

- User Method SayHi
- Logging
- LogMethod Advice
Target

What is a Target?

They are the object on which advices are applied. Spring AOP is implemented using runtime proxies so this object is always a proxied object. What is means is that a subclass is created at runtime where the target method is overridden and advices are included based on their configuration.
Proxy

What is a Proxy?

A Proxy is the object that is created or extended by adding an advice to a Target Object in a Join Point.
Weaving

what is a weaver?

Metaprogramming utility

It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime.
It doesn’t matter which language is used for specifying and implementing aspects and components?

- True
- False
Development Example
PROGRAMMING EXAMPLES
<bean id="customerService" class="example.com.CustomerService">
    <property name="name" value="Neill Giraldo" />
    <property name="url" value="www.neillgiraldo.com" />
</bean>

<bean id="customerService2" class="example.com.CustomerService">
    <property name="name" value="Ivan Vanegas" />
    <property name="url" value="www.ivanvanegas.org" />
</bean>

package example.com;

public class CustomerService {
    private String name;
    private String url;

    public void printName() {
        System.out.println("Customer name : " + this.name);
    }

    public void printURL() {
        System.out.println("Customer website : " + this.url);
    }
}

PROGRAMMING EXAMPLES
import java.lang.reflect.Method;
import example.com.CustomerService;
import org.springframework.aop.MethodBeforeAdvice;

public class CheckUrl implements MethodBeforeAdvice {
    @Override
    public void before(Method method, Object[] args, Object target) throws Throwable {
        if (method.getName().equals("printURL")) {
            CustomerService h = (CustomerService) target;
            String[] values = (h.getUrl()).split("\.");
            if (values[2].equals("com")) {
                System.out.println("Valid URL: " + h.getUrl());
            } else {
                System.out.println("Invalid URL: " + h.getUrl());
                h.setUrl("" considering this part is missing from the provided code snippet.
            }
        }
    }
}
public static void main(String[] args) {
    ApplicationContext appContext = new ClassPathXmlApplicationContext(
        new String[] { "Spring-Customer.xml" });

    String[] ids = {"customerServiceProxy","customerServiceProxy2"};

    for(String id: ids) {
        CustomerService cust = (CustomerService) appContext.getBean(id);

        System.out.println("*************************");
        cust.printName();
        System.out.println("*************************");
        cust.printURL();
        System.out.println("*************************");
        try {
            cust.printThrowException();
        } catch (Exception e) { e.printStackTrace();}
    }
}
<bean id="CheckUrlBean" class="advices.CheckUrl" />

<bean id="customerServiceProxy"
   class="org.springframework.aop.framework.ProxyFactoryBean">
   <property name="target" ref="customerService" />
   <property name="interceptorNames">
      <list>
         <value>CheckUrlBean</value>
      </list>
   </property>
</bean>

<bean id="customerServiceProxy2"
   class="org.springframework.aop.framework.ProxyFactoryBean">
   <property name="target" ref="customerService2" />
   <property name="interceptorNames">
      <list>
         <value>CheckUrlBean</value>
      </list>
   </property>
</bean>

PROGRAMMING EXAMPLES
Advantages & Disadvantages
Advantages

- High Modularity, easy coupling of components and aspects
- High quality software development, permitting sophisticated methodologies such as run-time system redesign

Disadvantages

- Cost of new technologies introduction on development processes
- Not yet known or accepted commercially
Criticism
Criticism

- Obscure Control Flow - Come From Statement.
  - Code not as easy to read
- Undermines
  - Code Structure
- Impedes
  - Code understandability
  - Independent Development
Conclusions
Conclusions

- AOP is a very new programming paradigm which challenges old-fashioned abstraction and design approaches for software development while providing an extension for clear modularization of functional and nonfunctional requirements concerns.

- AOP takes into consideration such important features of a system as data-details and execution flow of basic functionalities for achieving high separation of responsibilities but letting them, at the same time, be coupled interactively via the weavers technology language specifications.
Thank you

Programming Languages Course, 2017-1, Universidad Nacional de Colombia
Exercise Review
Bibliography

1. Programación Orientada a Aspectos, 2016, repositorio curso unal
   http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teoria/index.html

Other consulted websites:

5. http://includeblogh.blogspot.com.co/search?q=Programacion%C3%B3n+Orientada+a+Aspectos
6. https://es.slideshare.net/wfranck/programacion-orientada-a-aspectos-poa