Ingeniería del Software II

Seminario. Programación Orientada a Aspectos

Pablo Sánchez Barreiro

Dpto. de Matemáticas, Estadística y Computación

p.sanchez@unican.es

Este tema se publica bajo Licencia: Creative Commons BY-NC-SA 3.0
Table of Contents

1 Introduction: Why we need Aspect-Oriented Programming
2 Aspect-Oriented Programming
3 Current Challenges
4 What AOSD Is Not About
5 AOSD and Related Technologies
6 Conclusions
7 References
Crosscutting Concerns Drawbacks

- Base Functionality
- Access Control
- Fault Tolerance
- Encryption

Modules:
- Module A
- Module B
- Module C
- Module D
- Module E
Case Study: Apache Tomcat

XML parsing
Case Study: Apache Tomcat

URL pattern matching
Case Study: Apache Tomcat

Logging
Motivating example: Access Control
Motivating example: Access Control

public class Student {

    protected String religion;
    protected AccessControl authenticator = new AccessControl();

    public String getReligion() {

        String result = null;

        if (authenticator.getAccess()) {
            result = this.religion;
        } // if

        return result; // getReligion
    } // getReligion

} // Student

Crosscutting code
Motivating example: Access Control

```java
public List<String> getDiseases() {
    List<String> result = null;

    if (authenticator.getAccess()) {
        result = this.diseases;
    } // if

    return result; // getDiseases
}
```
Main Shortcoming of Crosscutting Concerns

Base Functionality
Access Control
Fault Tolerance
Encryption

Module A
Module B
Module C
Module D
Module E
Conclusions

Scattering
A concern appears spread over several software modules.

Tangling
Several concern appears software modules.

1. **Scattering:** Decreases encapsulation, (consequently decreases cohesion), *creates redundancies*, hinders reutilization of the crosscutting concern and increase coupling.

2. **Tangling:** Decreases cohesion, (consequently hampers reutilization), and increases coupling.
Concern-Oriented Metrics [Sant’Anna et al., 2007]

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDC</td>
<td>Concern Diffusion over Classes</td>
</tr>
<tr>
<td>CDO</td>
<td>Concern Diffusion over Operations</td>
</tr>
<tr>
<td>CIC</td>
<td>Class-Level Interlacing between Concerns</td>
</tr>
<tr>
<td>OOC</td>
<td>Operation-Level Overlapping between Concerns</td>
</tr>
<tr>
<td>ACC</td>
<td>Afferent Coupling between Classes</td>
</tr>
<tr>
<td>ECC</td>
<td>Efferent Coupling between Classes</td>
</tr>
<tr>
<td>LCC</td>
<td>Lack of Concern-Based Cohesion</td>
</tr>
</tbody>
</table>
Concern-Oriented Metrics [Sant’Anna et al., 2007]

```java
class AccessControl {
    +askForCredentials( out username : String, out password : String )
    +getAccess() : Boolean
    +getSelectiveAccess( String : String )
    -name : String
    -address : String
    -religion : String
    -diseases : String [0..*]
    +getName() : String
    +getAddress() : String
    +getReligion() : String
    +getDiseases() : String [0..*]
}
```

**Data Storage (base)**

<table>
<thead>
<tr>
<th></th>
<th>CDC</th>
<th>CDO</th>
<th>CIC</th>
<th>OOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AccessControl</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Student**

<table>
<thead>
<tr>
<th></th>
<th>ACC</th>
<th>ECC</th>
<th>LCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>AccessControl</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AccessControlView</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Table of Contents

1 Introduction: Why we need Aspect-Oriented Programming
2 Aspect-Oriented Programming
3 Current Challenges
4 What AOSD Is Not About
5 AOSD and Related Technologies
6 Conclusions
7 References
Aspect-Oriented Programming Principle
Aspect-Oriented Programming Principle

Module A

Joinpoint

Joinpoint

Module B

Joinpoint

Joinpoint
Aspect-Oriented Programming Principle

Fault Tolerance

Module A

Module B

Access Control

Encryption
Aspect-Oriented Programming Principle

- Fault Tolerance
- Access Control
- Encryption
- Pointcut

Module A

Module B
Weaving Process

Aspect-Oriented Weaving Process

Base Model → Aspect Weaver → Woven Base Model → Common Compiler → Compiled Code

+++

Pablo Sánchez (MATESCO)
Weaving Process

Weaver

Concern Identifier

Concern Identifier

Weaver
Kinds of Aspect-Oriented Weaving

Static Weaving  Aspects are woven at compile time [Kiczales et al., 2001].

Load-Time Weaving  Aspects are woven at class load-time [Laddad, 2001, Chiba, 2000].

Dynamic Weaving  Aspects can be woven and even unwoven at runtime [Pinto et al., 2005, Suvée et al., 2003]. It is particularly interesting for context-aware systems [Fuentes et al., 2009].
public aspect AccessControlAspect {

    pointcut guardedMethods():
        call (String Student.getReligion());

    String around() : guardedMethods() {

        AccessControl guard = new AccessControl();

        String returnValue = null;

        if (guard.getAccess()) {
            returnValue = proceed();
        } // if

        return returnValue;
    } // around guardedMethods
} // AccessControlAspect
A Clean Student Class

```java
public class Student {

    protected String religion;

    public String getReligion() {
        return this.religion;
    }

} // Student
```
public aspect QuantifiedAccessControlAspect {

    pointcut guardedMethods(): call(* Student.get*(..));

    Object around() : guardedMethods() {

        AccessControl guard = new AccessControl();

        Object returnValue = null;

        if (guard.getAccess()) {
            returnValue = proceed();
        }

        return returnValue;
    }
}

// QuantifiedAccessControlAspect
Annotation-based Quantified Pointcuts

```java
public class Student {

    protected String religion;

    @AccessControlRequired
    public String getReligion() {
        return this.religion;
    }

    } // getReligion

    } // Student
```
Annotation-based Quantified Pointcuts

```java
public aspect AnnotationBasedAspect {

    pointcut guardedMethods():
        call(@AccessControlRequired * *.*(..));

    Object around() : guardedMethods() {
        AccessControl guard = new AccessControl();
        Object returnValue = null;

        if (guard.getAccess()) {
            returnValue = proceed();
        } // if

        return returnValue;
    } // around guardedMethods

} // AnnotationBasedAspect
```
Some Basic & Free Reflection

```java
class AccessControl {
    // Access Control
    public int getAccess() { return 0; }
    public boolean getSelectiveAccess(String username, String password) { return false; }

    // Access Control View
    public String askForCredentials(String username, String password) {
        return null;
    }
}
```

```java
import java.util.List;

public class AccessControl {
    private String name;
    private String address;
    private String religion;
    private List<String> diseases;

    public AccessControl(String name, String address, String religion, List<String> diseases) {
        this.name = name;
        this.address = address;
        this.religion = religion;
        this.diseases = diseases;
    }

    public String getName() { return name; }
    public String getAddress() { return address; }
    public String getReligion() { return religion; }
    public List<String> getDiseases() { return diseases; }
}
```

```java
public class Student {
    private AccessControl accessControl;

    public Student(AccessControl accessControl) {
        this.accessControl = accessControl;
    }

    public List<String> askForCredentials(String username, String password) {
        return accessControl.askForCredentials(username, password);
    }
}
```
public aspect ReflectionAccessControlAspect {

  pointcut guardedMethods():
    call(String Student.getReligion());

  String around() : guardedMethods() {
    AccessControl guard = new AccessControl();
    String returnValue = null;
    if (guard.getSelectiveAccess(thisJoinPoint.getSignature().getName())) {
      returnValue = proceed();
    } // if
    return returnValue;
  } // around guardedMethods

} // ReflectionAccessControlAspect
Concern-Oriented Metrics Revisited

```
class AccessControlAspect

Student
- name : String
- address : String
- religion : String
- diseases : String [0..*]
+ getReligion() : String
+ getName() : String
+ getAddress() : String
+ getDiseases( diseases : String [0..*] )

AccessControl
+ getAccess() : Boolean
+ getSelectiveAccess( String : String )

AccessControlView
+ askForCredentials( out username : String, out password : String )
```

---

Data Storage (base)

<table>
<thead>
<tr>
<th></th>
<th>CDC</th>
<th>CDO</th>
<th>CIC</th>
<th>OOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Storage (base)</td>
<td>1 (1)</td>
<td>4 (4)</td>
<td>0 (1)</td>
<td>0 (1)</td>
</tr>
<tr>
<td>AccessControl</td>
<td>2 (2)</td>
<td>3 (5)</td>
<td>0 (1)</td>
<td>0 (1)</td>
</tr>
<tr>
<td>AccessControlView</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th></th>
<th>ACC</th>
<th>ECC</th>
<th>LCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>1 (0)</td>
<td>0 (1)</td>
<td>1 (2)</td>
</tr>
<tr>
<td>AccessControl</td>
<td>0 (1)</td>
<td>2 (1)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>AccessControlView</td>
<td>1 (1)</td>
<td>0 (0)</td>
<td>1 (1)</td>
</tr>
</tbody>
</table>
Table of Contents

1. Introduction: Why we need Aspect-Oriented Programming
2. Aspect-Oriented Programming
3. Current Challenges
4. What AOSD Is Not About
5. AOSD and Related Technologies
6. Conclusions
7. References
Current Hot Research Topics on AOSD

1. Semantic pointcuts [Ostermann et al., 2005].
2. Reusable aspects (e.g. Complex Transaction Management) [Kienzle and Gélineau, 2006].
3. Aspect interference/conflicts [Douence et al., 2002, Douence et al., 2004, Altahat et al., 2008].
4. Complex pointcuts [Allan et al., 2005].
5. Aspect Influence on Interfaces [Ostermann, 2008].
Current Hot Research Topics on AOSD

Diagram showing the interaction between different components such as `checkOut(..)`, `newCreditCard(..)`, and `approveCredit(..)` involving `ShoppingCart`, `CreditCard`, and `WebServiceBank`.

- `checkOut(..)` and `newCreditCard(..)` are connected to `ShoppingCart`.
- `approveCredit(..)` involves `CreditCard` and `WebServiceBank`.
- The diagram includes a call behavior `calculateDiscount` and `minus` to manipulate prices.

Terms and Conditions of Seasonal Rebate:

- Price calculation and approval of credit cards are part of the seasonal rebate terms and conditions.
Table of Contents

1. Introduction: Why we need Aspect-Oriented Programming
2. Aspect-Oriented Programming
3. Current Challenges
4. What AOSD Is Not about
5. AOSD and Related Technologies
6. Conclusions
7. References
What AOSD is not about

AOSD is not View-Based Software Development

class addItem

UserGUI IShop ShoppingCart

+ addItem (book :Book, amount :int)

Books + books 0..*

sd addItem

UserGUI ShoppingCart

class checkOut

UserGUI IShop ShoppingCart

+ checkOut ()

IDelivery Bank

sd checkOut

UserGUI ShoppingCart Delivery Bank

checkDelivery (address =-, weight =-, size =-)

approveCredit (ccNumber =-, amount =-)

merge

Pablo Sánchez (MATESCO)
AOSD is not Feature-Oriented Software Development

What AOSD is not about

AOSD is not Feature-Oriented Software Development
Table of Contents

1. Introduction: Why we need Aspect-Oriented Programming
2. Aspect-Oriented Programming
3. Current Challenges
4. What AOSD Is Not About
5. AOSD and Related Technologies
6. Conclusions
7. References
AOSD and Related Technology

1. Component platforms/Application Servers/Container Technology
   - Component platforms offer a set of fixed services.
   - Configuration of each service is often different.
   - Component platforms are often heavier.

2. Reflection: AOP is reflection for human beings.

3. Design Patterns, Dynamic Proxies: The aspect-oriented weaver does it for you.
AOSD in Industry

1. Spring framework for Enterprise Java.
2. SpringSource dm Server, SpringSource tc Server, Spring Roo y Magma.
3. GlassBox, Perf4J, Contract4J, JXInsight, MaintainJ.
4. Websphere, MySQL, JBoss, Oracle Toplink, J2ME [Rashid et al., 2010].
5. Siemens Healthcare [Rashid et al., 2010].
6. Motorola Weavr [Cottenier et al., 2007].
7. Oficina Virtual Fundación Retevisión [Pinto et al., 2005]
8. Digital Publishing [de Beeck et al., 2009].
9. Isis Project (Stop Pedophilia).
Table of Contents

1. Introduction: Why we need Aspect-Oriented Programming
2. Aspect-Oriented Programming
3. Current Challenges
4. What AOSD Is Not About
5. AOSD and Related Technologies
6. Conclusions
7. References
Conclusions

Aspect-Oriented Software Development

Aspect-Oriented Software Development can help to improve modularization of crosscutting concerns, easing maintenance and evolution.

Access Control Code:
http://personales.unican.es/sanchezbp/teaching/ao/accesscontrol.zip

More Information:
http://www.aosd-europe.net
Table of Contents

1. Introduction: Why we need Aspect-Oriented Programming
2. Aspect-Oriented Programming
3. Current Challenges
4. What AOSD Is Not About
5. AOSD and Related Technologies
6. Conclusions
7. References
Adding trace matching with free variables to AspectJ.

Applying Critical Pair Analysis in Graph Transformation Systems to Detect Syntactic Aspect Interaction in UML State Diagrams.
In Proc. of the 20th Int. Conference on Software Engineering & Knowledge Engineering (SEKE), pages 905–911, San Francisco (California, USA).


References VI


Java World,


Questions ?