Comparative Analysis Of Strategies For Applying AOP On Legacy Code

Forth Turkish Aspect-Oriented Software Development Workshop (Bilkent Ankara 12/2009)

Duygu Sarıkaya
Can Ufuk Hantaş
Dilek Demirbaş
Outline

- Introduction
- Case Study
- Strategies
  - Direct Application Of Aspects To Legacy Code Without Refactoring
  - Dealing crosscutting concerns by Object-Oriented re-factoring without AOP
  - Applying AOP After Object-Oriented Re-factoring
- Conclusion
Problem Definition

- Object Oriented Programming
- Design Patterns
- Aspect Oriented Programming
Problem Definition

- Problems with Existing Code
The Goal

- Investigate the Usage of AOP in existing code.
Our Approach

- Use Aspect Oriented Approach
  - Not only as complementary to the shortages of OOP programming
  - But also solely to create solutions in a fast manner
    - Which saves
      - Time
      - Effort
      - Cost
Case Study

- E-Book Manager Application
Case Study: E-Book Manager Application

- Class Diagram
Our Strategy

- Direct Application Of Aspects To Legacy Code Without Refactoring
  - Existing Concerns
  - New Concerns

- Dealing crosscutting concerns by Object-Oriented re-factoring without AOP
  - Existing Concerns
  - New Concerns

- Applying AOP After Object-Oriented Re-factoring
  - Existing Concerns
  - New Concerns
Our Strategies: Direct Application Of Aspects To Legacy Code Without Refactoring

- **Existing Concerns**
  - Detail Storage
  - Checkout
  - Web Service

- **New Concerns**
  - Internationalization
  - Look&Feel
Existing Concerns:
Detail Storage

- By Using Aspect Oriented Approach
  - 3 point cuts – 3 advices
    - Save Details
      - Saving Ebook objects with file streams
    - Load Details
      - Loading files and casting them to Ebook objects
    - Copy File
      - Copying detail storage files along with Ebook files
Existing Concerns
: Detail Storage

public aspect DetailStorage {

pointcut saveBook(Ebook book, int i,MainFrame mf) : call(* MainFrame.SaveBook(Ebook , int)) && args(book,i) && target(mf);

pointcut loadBook(String fileName, BookTableDataModel btdm) : call(* BookTableDataModel.breakFileNam e(String)) && args(fileName) && target(btdm);

pointcut copyFile(String src, String dst): call(* Utility.copyfile(..)) && args(src, dst) && this(FileCopyCheckOut);

declare parents: Ebook implements Serializable; // use Serializable interface to store book information

// Advice for overriding method calls to breakFileName method to get boot information from file name and create book object

Ebook around(String s, BookTableDataModel btdm) : loadBook(s, btdm){

... 

}

...

...
Existing Concerns: Checkout

- By Using Aspect Oriented Approach
  - Menu integration
  - Implementation
public aspect CheckOutManager {

  pointcut checkOut(MainFrame mf) : call(* MainFrame.CheckoutFiles()) && target(mf);
  pointcut checkOutFileMenuFunction(MainFrame mf): call(* MainFrame.mnuCheckoutActionPerformed(*)) && target(mf);
  pointcut menuCreation(MainFrame mf): call(* MainFrame.initComponents()) && this(mf);

  //override checkOut function so that it will work according to selected check out manager
  void around(MainFrame mf) : checkOut(mf) {
      ...
  }

  //Create send-by-email menu
  after(final MainFrame mf): menuCreation(mf) {
      ...
  }

  //Set manager to fileCopyManager before menu action performed
  before(MainFrame mf): checkOutFileMenuFunction(mf) {
      ...
  }
}
Existing Concerns
:Web Service

:Aspect code of Web Service
Our Strategies: New Concerns

- Dealing with crosscutting concerns that come with adding related functionalities
  - Internationalization
  - Look and Feel
  - Agent Based Communication
New Concerns: Internationalization

- Swing and AWT elements
  - `SetText()` method
  - Language files
public aspect internationalization
{
pointcut setText(String s, JComponent comp): call(* *.setText(String)) && args(s) && target(comp) && !this(internationalization);
void around(String s, JComponent comp): setText(s, comp) {
// some variable declarations
...
locale = new Locale("tr","TR");
rb = ResourceBundle.getBundle("Messages", locale);
// getting locale value according to keys
...
if(comp instanceof JMenuItem){
JMenuItem jm=(JMenuItem)comp;
jm.setText(s);
}
// check for other types
...
}
New Concerns: Look and Feel

- Swing look & feels
- OS dependent visual style
- Component type specific style
New Concerns: Look and Feel

```java
public aspect lookandfeel {
  pointcut catchButton(Jbutton j): set(JButton *.* ) && args(j);
  after(Jbutton j):catchButton(j){
    int osType =DetailStorage.osType();
    switch (osType){
      case 0://Unknown OS
        break;
      case 1://Windows
        j.setUI(new javax.swing.plaf.basic.BasicButtonUI());
        break;
      case 2://Mac
        ...
        break;
      case 3://Unix, Linux
        j.setUI(new javax.swing.plaf.metal.MetalButtonUI()));
        break;
    }
  }
}
```
New Concerns: Agent Based Communication

*Figure 1: Containers and Platforms*
New Concerns: Agent Based Communication

```java
public aspect DistributedSellerAgent {

    pointcut loadFilesIntoModel(BookTableDataModel btdm) : call(*
        BookTableDataModel.loadFilesIntoModel(..)) && target(btdm);

    after(BookTableDataModel btdm) : loadFilesIntoModel(btdm){
        ...
    }
}
```
Our Strategies: Dealing crosscutting concerns by Object-Oriented re-factoring without AOP

- Existing Concerns
  - Detail Storage
  - Checkout
  - Web Service

- New Concerns
  - Internationalization
  - Look&Feel
Existing Concerns: Detail Storage

- By rearranging the Code
Existing Concerns: Checkout

- By rearranging the Code
New Concerns

- Internationalization
  - require many changes to be done on the existing code
- Look & Feel
- Agent Based Communication
Our Strategies: Applying AOP After Object-Oriented Re-factoring

- **Existing Concerns**
  - Detail Storage
  - Checkout
  - Web Service

- **New Concerns**
  - Internationalization
  - Look&Feel
Existing Concerns: Detail Storage

- By rearranging the code
Existing Concerns: Checkout

- Refactorization strategy
- Adding new functionality to menu
Existing Concerns: Web Service

- Changed the code in place
- Could be done in a separate class
Our Strategies: New Concerns

- Internationalization
- Look&Feel
- Agent Based Communication
Direct application of aspects to legacy code without re-factoring
Discussion

- Dealing crosscutting concerns by Object-Oriented re-factoring without AOP
Discussion

Applying AOP after Object-Oriented refactoring
## Discussion

<table>
<thead>
<tr>
<th></th>
<th>Existing Concerns</th>
<th>New Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Directly Applying AOP</strong></td>
<td>• Easy to implement</td>
<td>• Easy to implement</td>
</tr>
<tr>
<td></td>
<td>① Does not require understanding all parts of code</td>
<td>① Do not require wide understanding on existing code.</td>
</tr>
<tr>
<td></td>
<td>② No need to make change on existing code</td>
<td>② Could be applied to the project easily</td>
</tr>
<tr>
<td></td>
<td></td>
<td>③ Improves extendibility for future aspectual concerns.</td>
</tr>
<tr>
<td><strong>Re-factorization</strong></td>
<td>① Decreases crosscutting concerns</td>
<td>① For some concerns not possible to solve by re-factorization</td>
</tr>
<tr>
<td></td>
<td>② In some cases may not be possible.</td>
<td>② Requires hard coding within existing classes</td>
</tr>
<tr>
<td></td>
<td>③ Hard to understand existing scattered code</td>
<td></td>
</tr>
<tr>
<td></td>
<td>④ Time consuming</td>
<td></td>
</tr>
<tr>
<td></td>
<td>⑤ Increases code reuse-ability</td>
<td></td>
</tr>
<tr>
<td><strong>Re-factorization and AOP</strong></td>
<td>① Could deal with crosscutting concerns which OOP re-factorization could not solve itself</td>
<td>① Prevents crosscutting concerns</td>
</tr>
<tr>
<td></td>
<td>② Improves modularity</td>
<td>② Improves modularity</td>
</tr>
</tbody>
</table>
Demo

Thank You...
Questions are Welcomed...