AjMutator: a Tool for the Mutation Analysis of Pointcut Descriptors

Romain Delamare\textsuperscript{1} \quad Benoit Baudry\textsuperscript{1} \quad Yves Le Traon\textsuperscript{2}

\textsuperscript{1}IRISA / INRIA
Rennes, France

\textsuperscript{2}Télécom Bretagne
Rennes, France

April 5\textsuperscript{th} 2009 \quad Mutation 2009 \quad Denver
Aspect Oriented Programming

• The crosscutting concerns are separated from the core concern

• Aspect

• Advice

• Pointcut Descriptor (PCD)

• Describes a set of joinpoints
Aspect Oriented Programming: example

**Bank**

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ login(String, String): boolean</td>
</tr>
<tr>
<td>+ deleteAccount(int): boolean</td>
</tr>
<tr>
<td>+ createAccount(): Account</td>
</tr>
<tr>
<td>+ getAccount(int): Account</td>
</tr>
</tbody>
</table>

**Account**

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>- balance</td>
</tr>
<tr>
<td>+ withdraw(int): boolean</td>
</tr>
<tr>
<td>+ deposit(int): void</td>
</tr>
</tbody>
</table>

accounts
Aspect Oriented Programming: example

```
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```
Aspect Oriented Programming: example

```
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```

Bank
- login(String,String): boolean
- deleteAccount(int): boolean
- createAccount(): Account
- getAccount(int): Account

Account
- balance
- withdraw(int): boolean
- deposit(int): void

accounts
- *
Aspect Oriented Programming: example

```java
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))
    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```
Aspect Oriented Programming: example

```java
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```
Aspect Oriented Programming: example

Bank

+ login(String,String): boolean
+ deleteAccount(int): boolean
+ createAccount(): Account
+ getAccount(int): Account

Account

- balance
+ withdraw(int): boolean
+ deposit(int): void

money > 0 ?

withdraw

money > 0 ?

withdraw

no
Aspect Oriented Programming: example

<table>
<thead>
<tr>
<th>Bank</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ login(String,String): boolean</td>
<td>- balance</td>
</tr>
<tr>
<td>+ deleteAccount(int): boolean</td>
<td>+ withdraw(int): boolean</td>
</tr>
<tr>
<td>+ createAccount(): Account</td>
<td>+ deposit(int): void</td>
</tr>
<tr>
<td>+ getAccount(int): Account</td>
<td></td>
</tr>
</tbody>
</table>

- authorized?  
- money > 0?  
- withdraw

- no
- yes
- yes
- no
Classes of Faults in the Pointcut Descriptor

Intended

Matched
Classes of Faults in the Pointcut Descriptor

correct PCD
Classes of Faults in the Pointcut Descriptor

- correct PCD
- neglected joinpoints

Intended  Matched
Classes of Faults in the Pointcut Descriptor

- correct PCD
- unintended joinpoints
- matched joinpoints

Intended - Matched
Classes of Faults in the Pointcut Descriptor

- correct PCD
- both neglected and unintended
- neglected joinpoints
- unintended joinpoints

Intended  Matched
Classes of Faults in the Pointcut Descriptor

- correct PCD
- class 1
- class 2
- class 3

Intended

Matched
Mutant Pointcut Descriptor

- A PCD where a fault has been inserted
  - Selects a different set of joinpoints
- Equivalent mutant
  - Mutant that matches the same set of joinpoint
Mutant Pointcut Descriptor

• A PCD where a fault has been inserted
  • Selects a different set of joinpoints
• Equivalent mutant
  • Mutant that matches the same set of joinpoint
• Equivalent mutants can be detected statically
AjMutator: Overview

Mutant Generation → Mutant Compilation → Test Cases Running
AjMutator: Overview

- Mutant Generation
  - PCD parsing

- Mutant Compilation

- Test Cases Running
AjMutator: Overview

- Mutant Generation
  - PCD parsing
  - AST for each PCD

- Mutant Compilation

- Test Cases Running
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

Test Cases Running
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation

Test Cases Running

• PCD parsing
• AST for each PCD
• Fault insertion by the mutation operators
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation
- Compilation

Test Cases Running
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation
- Compilation
  - A jar file for each mutant

Test Cases Running
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation
- Compilation
- A jar file for each mutant
- Classification

Test Cases Running
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation
- Compilation
  - A jar file for each mutant
- Classification
- Selection

Test Cases Running
AjMutator: Overview

**Mutant Generation**
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

**Mutant Compilation**
- Compilation
  - A jar file for each mutant
- Classification
- Selection

**Test Cases Running**
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation
- Compilation
- A jar file for each mutant
- Classification
- Selection

Test Cases Running
- Test cases are executed on each mutant system
AjMutator: Overview

Mutant Generation
- PCD parsing
- AST for each PCD
- Fault insertion by the mutation operators

Mutant Compilation
- Compilation
- A jar file for each mutant
- Classification
- Selection

Test Cases Running
- Test cases are executed on each mutant system
- Mutation score for the test suite
Mutant Generation

• An Abstract Syntax Tree (AST) for each PCD
• Mutation operators from [Ferrari et al., ICST’08]
• Implemented as AST visitors
  • Inserts the fault by modifying the AST
  • The AST is pretty-printed in a mutant source file
• New operators can added easily
# Mutation Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCCC</td>
<td>Replaces a <code>cflow</code> by a <code>cflowbelow</code>, or the contrary</td>
</tr>
<tr>
<td>PCCE</td>
<td>Replaces a <code>call</code> by an <code>execution</code>, or the contrary</td>
</tr>
<tr>
<td>PCGS</td>
<td>Replaces a <code>get</code> by a <code>set</code>, or the contrary</td>
</tr>
<tr>
<td>PCLO</td>
<td>Changes the logical operators in a composition of PCDs</td>
</tr>
<tr>
<td>PCTT</td>
<td>Replaces a <code>this</code> by a <code>target</code>, or the contrary</td>
</tr>
<tr>
<td>POEC</td>
<td>Adds, removes or changes throwing clauses</td>
</tr>
<tr>
<td>POPL</td>
<td>Changes the parameter list</td>
</tr>
<tr>
<td>PSWR</td>
<td>Removes wildcards</td>
</tr>
<tr>
<td>PWAR</td>
<td>Removes annotation from type, field or method patterns</td>
</tr>
<tr>
<td>PWIW</td>
<td>Adds wildcards</td>
</tr>
</tbody>
</table>
Mutant Generation: problem

• What we want: mutants PCDs selecting different joinpoints
  
• What we do: modify the declaration of the PCD

• Problem:
  
  • Several different declarations can describe the same PCD
  
  • Thus we can have an equivalent mutant.
Mutant Compilation

- Each mutant is compiled
- If the compilation does not fail, the mutant is classified
  - Automatic classification, using the previous classification
- A selection of the mutant is made, depending on their class
  - The mutation analysis only considers selected mutants
Automatic Classification and Selection of the Mutants

compiles?
Automatic Classification and Selection of the Mutants

compiles?

no

non compilable
Automatic Classification and Selection of the Mutants

Diagram:
- If `compiles?` is yes, go to `neglected?`.
- If `neglected?` is yes, go to `unintended?`.
- If `unintended?` is yes, the class is 1.
- If `compiles?` is no or any of the previous questions are no, the mutants are non-compilable.
Automatic Classification and Selection of the Mutants

- **Compiles?**
  - Yes → **Neglected?**
    - Yes → **Unintended?**
      - Yes → **Class 1 both**
      - No → **Class 2 neglected**
    - No → **Non compilable**

- No
Automatic Classification and Selection of the Mutants

- **compiles?**
  - yes: **neglected?**
    - yes: **unintended?**
      - yes: class 1 both
      - no: class 2 neglected
    - no: non compilable
  - no: unintended
    - yes: class 3 unintended
    - no: non compilable
Automatic Classification and Selection of the Mutants

- If it compiles?
  - Yes: If neglected?
    - Yes: If unintended?
      - Yes: Class 1 both
      - No: Class 2 neglected
    - No: Class 3 unintended
  - No: Non compilable
- No: Equivalent
Automatic Classification and Selection of the Mutants

-compiles?

neglected?

unintended?

class 1 both

class 2 neglected

class 3 unintended

non compilable

equivalent

Selected
Automatic Classification and Selection of the Mutants

1. **Compiles?**
   - Yes → **Neglected?**
   - No → **Unintended?**

2. **Neglected?**
   - Yes → **Unintended?**
   - No → **Class 2, Neglected**

3. **Unintended?**
   - Yes → **Class 1, Both**
   - No → **Class 3, Unintended**

4. **Uncompilable** → **Not Selected**

5. **Equivalent** → **Selected**
Static and Dynamic PCDs

• All PCDs have a static part
  • Worst case: all the joinpoints of the program
• Some PCDs have a dynamic part (Dynamic PCDs)
  • At runtime the dynamic part decides whether the advice is executed or not (restriction of the joinpoints)
  • At compile time the set of joinpoints matched by a dynamic PCD can only be over-approximated
Dynamic PCD: example

<table>
<thead>
<tr>
<th>Bank</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ login(String,String): boolean</td>
<td>- balance</td>
</tr>
<tr>
<td>+ deleteAccount(int): boolean</td>
<td>+ withdraw(int): boolean</td>
</tr>
<tr>
<td>+ createAccount(): Account</td>
<td>+ deposit(int): void</td>
</tr>
<tr>
<td>+ getAccount(int): Account</td>
<td></td>
</tr>
</tbody>
</table>

accounts

*
Dynamic PCD: example

**pointcut** controlledAccess(): get(Account.balance) && cflow(execution(* Auction.withdraw(int))))
Dynamic PCD: example

<table>
<thead>
<tr>
<th>Bank</th>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ login(String,String): boolean</td>
<td>- balance</td>
</tr>
<tr>
<td>+ deleteAccount(int): boolean</td>
<td>+ withdraw(int): boolean</td>
</tr>
<tr>
<td>+ createAccount(): Account</td>
<td>+ deposit(int): void</td>
</tr>
<tr>
<td>+ getAccount(int): Account</td>
<td></td>
</tr>
</tbody>
</table>

pointcut controlledAccess(): get(Account.balance) &&
cflow(execution(* Auction.withdraw(int)))

public boolean withdraw(int amount) {
    if(balance>0) {
        // ...
    }
}
Dynamic PCD: example

Bank
+ login(String,String): boolean
+ deleteAccount(int): boolean
+ createAccount(): Account
+ getAccount(int): Account

Account
- balance
+ withdraw(int): boolean
+ deposit(int): void

pointcut controlledAccess(): get(Account.balance) &&
cflow(execution(* Auction.withdraw(int)))

public boolean withdraw(int amount) {
  if(balance>0) {
    // ...
  }
}

public void deposit(int amount) {
  if(balance>0) {
    // ...
  }
}
Classification and Selection with Dynamic PCDs

compiles? yes

neglected? yes

unintended? yes

class 1 both

neglected? no

class 2 neglected

unintended? yes

class 3 unintended

unintended? no

equivalent

non compilable

no
Classification and Selection with Dynamic PCDs

- Compiles?
  - Yes → Unintended?
    - Yes → Class 1 both
    - No → Neglected?
      - Yes → Class 2 neglected
      - No → Unintended?
        - Yes → Class 3 unintended
        - No → Dynamic?
          - Yes → Unknown
          - No → Equivalent
            - Non compilable

Classification and Selection with Dynamic PCDs

- Compiles?
  - Yes
    - Neglected?
      - Yes
        - Unintended?
          - Yes
            - Class 1 both
          - No
            - Class 2 neglected
        - No
          - Unintended?
            - No
              - Dynamic?
                - Yes
                  - Unknown
                - No
                  - Equivalent
          - Yes
            - Class 3 unintended
    - No
      - Intended?
        - Yes
          - Class 1 both
        - No
          - Class 2 neglected

Not Selected
- Non compilable
- Equivalent

Selected
Execution of the Test Suite

• The test suite is executed on each mutant system
  • JUnit test suite
    • All tests pass on the original system
    • A mutant is killed if at least one test case fails
      • Qualification of the JUnit oracle
    • A mutation score for the test suite
Conclusion

• AjMutator, a tool for the mutation analysis of PCDs
  • Operators insert faults in the PCDs
  • Mutant are compiled, classified, and selected automatically
    • Automatic detection of the equivalent mutant in most cases
  • Execution of a Test Suite

• http://www.irisa.fr/triskell/softwares-fr/protos/AjMutator/
Evaluation on HealthWatcher

<table>
<thead>
<tr>
<th>Class</th>
<th>Number of Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (both)</td>
<td>55</td>
</tr>
<tr>
<td>2 (neglected)</td>
<td>50</td>
</tr>
<tr>
<td>3 (unintended)</td>
<td>129</td>
</tr>
<tr>
<td>unknown</td>
<td>65</td>
</tr>
<tr>
<td><strong>Total Selected</strong></td>
<td><strong>299</strong></td>
</tr>
<tr>
<td>Equivalent</td>
<td>296</td>
</tr>
<tr>
<td>Non-Compilable</td>
<td>90</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>685</strong></td>
</tr>
</tbody>
</table>