AOP and the Antinomy of the Liar

F. Forster  F. Steimann

Programming Systems
Department of Computer Science
University of Hagen

FOAL 2006
Outline

1. Famous Antinomies

2. Great Escapes

3. A Standard AOP Application
   - Tracing Problem
   - Workaround
   - Solution

4. Conclusion and Outlook
Theorem One

"Theorem Two" is true
Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false
Famous Antinomies - 1

Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation
false("Theorem Two" is true)
**Famous Antinomies - 1**

**Theorem One**
"Theorem Two" is true

**Theorem Two**
"Theorem One" is false

**Interpretation**

1. \( \text{false("Theorem Two" is true)} \Rightarrow \text{"Theorem Two" is false} \)
Famous Antinomies - 1

Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation
1. \(\text{false("Theorem Two" is true)} \Rightarrow "Theorem Two" \text{ is false}\)
2. \(\text{false("Theorem One" is false)}\)
Famous Antinomies - 1

Theorem One

"Theorem Two" is true

Theorem Two

"Theorem One" is false

Interpretation

1. \( \text{false}(" \text{Theorem Two} \) \) is true \( \Rightarrow \) "Theorem Two" is false
2. \( \text{false}(" \text{Theorem One} \) \) is false \( \Rightarrow \) "Theorem One" is true
Famous Antinomies - 1

Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation
1. false("Theorem Two" is true) ⇒ "Theorem Two" is false
2. false("Theorem One" is false) ⇒ "Theorem One" is true
3. true("Theorem Two" is true)
Famous Antinomies - 1

Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation
1. \( \text{false("Theorem Two" is true)} \Rightarrow \text{"Theorem Two" is false} \)
2. \( \text{false("Theorem One" is false)} \Rightarrow \text{"Theorem One" is true} \)
3. \( \text{true("Theorem Two" is true)} \Rightarrow \text{"Theorem Two" is true} \)
Theorem One

"Theorem Two" is true

Theorem Two

"Theorem One" is false

Interpretation

1. \( \text{false("Theorem Two" is true)} \Rightarrow \text{"Theorem Two" is false} \)
2. \( \text{false("Theorem One" is false)} \Rightarrow \text{"Theorem One" is true} \)
3. \( \text{true("Theorem Two" is true)} \Rightarrow \text{"Theorem Two" is true} \)
4. \( \text{true("Theorem One" is false)} \)
Famous Antinomies - 1

Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation

1. false("Theorem Two" is true) ⇒ "Theorem Two" is false
2. false("Theorem One" is false) ⇒ "Theorem One" is true
3. true("Theorem Two" is true) ⇒ "Theorem Two" is true
4. true("Theorem One" is false) ⇒ "Theorem One" is false
Famous Antinomies - 1

Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation

1. false("Theorem Two" is true) ⇒ "Theorem Two" is false
2. false("Theorem One" is false) ⇒ "Theorem One" is true
3. true("Theorem Two" is true) ⇒ "Theorem Two" is true
4. true("Theorem One" is false) ⇒ "Theorem One" is false
5. false("Theorem Two" is true)
Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation

1. false("Theorem Two" is true) ⇒ "Theorem Two" is false
2. false("Theorem One" is false) ⇒ "Theorem One" is true
3. true("Theorem Two" is true) ⇒ "Theorem Two" is true
4. true("Theorem One" is false) ⇒ "Theorem One" is false
5. false("Theorem Two" is true) ⇒ "Theorem Two" is false
Famous Antinomies - 1

Theorem One
"Theorem Two" is true

Theorem Two
"Theorem One" is false

Interpretation

1. false("Theorem Two" is true) ⇒ "Theorem Two" is false
2. false("Theorem One" is false) ⇒ "Theorem One" is true
3. true("Theorem Two" is true) ⇒ "Theorem Two" is true
4. true("Theorem One" is false) ⇒ "Theorem One" is false
5. false("Theorem Two" is true) ⇒ "Theorem Two" is false
6. repeat indefinitely.
Aspect One

public aspect S1 {
    void around(): adviceexecution() && within(S2) {
        proceed();
    }
}

Aspect Two

public aspect S2 {
    void around(): adviceexecution() && within(S1) {
    }
}
Russell’s Example

The Class of all those classes which are not members of themselves.
Russell’s Example

The Class of all those classes which are not members of themselves.

\[ M = \{ X | X \notin X \} \]
Russell’s Example

The Class of all those classes which are not members of themselves.

\[ M = \{ X \mid X \notin X \} \]

Assuming \( M \in M \)

\( M \in M \) contradicts the characteristic function of \( M \).
Russell’s Example

The Class of all those classes which are not members of themselves.

\[ M = \{X | X \notin X\} \]

Assuming \( M \in M \)

\( M \in M \) contradicts the characteristic function of \( M \Rightarrow M \notin M \)
Russell’s Example
The Class of all those classes which are not members of themselves.

\[ M = \{X | X \not\in X \} \]

Assuming \( M \in M \)
\( M \in M \) contradicts the characteristic function of \( M \Rightarrow M \not\in M \)

Assuming \( M \not\in M \)
\( M \not\in M \) fulfills the characteristic function of \( M \)
Russell’s Example

The Class of all those classes which are not members of themselves.

\[ M = \{ X | X \notin X \} \]

Assuming \( M \in M \)

\( M \in M \) contradicts the characteristic function of \( M \Rightarrow M \notin M \)

Assuming \( M \notin M \)

\( M \notin M \) fulfills the characteristic function of \( M \Rightarrow M \in M \)
Russell’s Solution

Whatever involves all of a collection must not be one of the collection.
Russell’s Solution
Whatever involves all of a collection must not be one of the collection.

Theory of Types
Whatever contains a variable must not be a possible value of that variable.
Russell’s Solution
Whatever involves all of a collection must not be one of the collection.

Theory of Types
Whatever contains a variable must not be a possible value of that variable.

Theory of Types for AspectJ
\[
someadvice_1(\{jp_1, jp_2, \ldots\}) \Rightarrow \text{joinpoints in } someadvice_1 \notin \{jp_1, jp_2, \ldots\}
\]
Task

Trace all Method execution and Advice execution.
Task

Trace all Method execution and Advice execution.

First Go

```java
public aspect Tracing {
    void around(): adviceexecution()||execution (* *(..)){
        System.out.println("Entering:" + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
```
**Task**

Trace all Method execution and Advice execution.

**First Go**

```java
public aspect Tracing {
    void around(): adviceexecution() || execution (* *(*(.))){
        System.out.println("Entering:" + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
```

**Problem**

around advice is an adviceexecution() joinpoint!
Task

Trace all Method execution and Advice execution.

First Go

```java
public aspect Tracing {
    void around(): adviceexecution() || execution (* *(..)){
        System.out.println("Entering:" + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
```

Problem

around advice is an adviceexecution() joinpoint! Theory of Types
public aspect Tracing {
    pointcut guard(): (adviceexecution() || execution (* *(..))) && within(Tracing);
    void around(): (adviceexecution() || execution (* *(..))) && !cflow(guard()) {
        System.out.println("Entering: " + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
public aspect Tracing {
    pointcut guard(): (adviceexecution() || execution (* *(..))) && within(Tracing);
    void around(): (adviceexecution() || execution (* *(..))) && !cflow(guard()) {
        System.out.println("Entering:" + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
Tracing - 2

Workaround

```java
public aspect Tracing {
    pointcut guard(): (adviceexecution()
        ||execution (* *(..))) && within(Tracing);
    void around(): (adviceexecution()
        ||execution (* *(..))) && !cflow(guard()) {
        System.out.println("Entering: " + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
```

Problem

Very verbose, error-prone,
Workaround

public aspect Tracing {
    pointcut guard(): (adviceexecution() || execution (* *(..))) && within(Tracing);
    void around(): (adviceexecution() || execution (* *(..))) && !cflow(guard()) {
        System.out.println("Entering: " + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}

Problem

Very verbose, error-prone, redundancy,
Workaround

```java
public aspect Tracing {
    pointcut guard(): (adviceexecution()
        ||execution (* *(..))) && within(Tracing);
    void around(): (adviceexecution()
        ||execution (* *(..))) && !cflow(guard()) {
        System.out.println("Entering:" + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
```

Problem

Very verbose, error-prone, redundancy, runtime check.
Using Russell’s Theory of Types for AOP means:
Using Russell’s Theory of Types for AOP means:

- An aspect is of higher type level than base code.
Using Russell’s Theory of Types for AOP means:

- An aspect is of higher type level than base code.
- An aspect advising another aspect is of higher level than the advised aspect.
Using Russell’s Theory of Types for AOP means:

- An aspect is of higher type level than base code.
- An aspect advising another aspect is of higher level than the advised aspect.
- Pointcuts in an aspect only select joinpoints in the scope of aspects/classes of lower level.
Using Russell’s Theory of Types for AOP means:

- An aspect is of higher type level than base code.
- An aspect advising another aspect is of higher level than the advised aspect.
- Pointcuts in an aspect only select joinpoints in the scope of aspects/classes of lower level.
- Syntactical constructs to distinguish levels.
Using Russell’s Theory of Types for AOP means:

- An aspect is of higher type level than base code.
- An aspect advising another aspect is of higher level than the advised aspect.
- Pointcuts in an aspect only select joinpoints in the scope of aspects/classes of lower level.
- Syntactical constructs to distinguish levels.

Solution Preview

```java
public meta aspect Tracing {
    void around(): adviceexecution()||execution (* *(..)) {
        System.out.println("Entering:" + thisJoinPoint);
        proceed();
        System.out.println("Leaving: " + thisJoinPoint);
    }
}
```
### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>class...</td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The keyword *meta*

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td><code>no pointcuts</code></td>
</tr>
</tbody>
</table>
### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td><code>no pointcuts</code></td>
</tr>
</tbody>
</table>
### New Syntax and Semantics

#### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td>aspect...</td>
<td><code>&lt;pc&gt;</code> except <code>adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td>class...</td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>
### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta aspect...</code></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td><code>&lt;pc&gt;</code> except <code>adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>

Forster, Steimann (University of Hagen)
## New Syntax and Semantics

### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta aspect...</code></td>
<td><code>adviceexecution()</code>,</td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td><code>&lt; pc &gt;</code> except <code>adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td><code>no pointcuts</code></td>
</tr>
</tbody>
</table>

*Forster, Steimann (University of Hagen) AOP and the Antinomy of the Liar FOAL 2006*
### New Syntax and Semantics

#### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta aspect...</code></td>
<td><code>adviceexecution()</code>, <code>meta &lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td><code>&lt; pc &gt; except adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td><code>no pointcuts</code></td>
</tr>
</tbody>
</table>
### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta aspect...</code></td>
<td><code>adviceexecution()</code>, <code>meta &lt; pc &gt; and &lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td><code>&lt; pc &gt; except adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>
The keyword *meta*

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td>meta(^2) aspect...</td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td>meta aspect...</td>
<td>adviceexecution(), meta (&lt; pc &gt;) and (&lt; pc &gt;)</td>
</tr>
<tr>
<td>Level 1</td>
<td>aspect...</td>
<td>(&lt; pc &gt;) except adviceexecution()</td>
</tr>
<tr>
<td>Level 0</td>
<td>class...</td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>
New Syntax and Semantics

The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td><code>meta^2 aspect...</code></td>
<td><code>meta adviceexecution()</code>,</td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta aspect...</code></td>
<td><code>adviceexecution()</code>, <code>meta &lt; pc &gt;</code> and <code>&lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td><code>&lt; pc &gt; except adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td><code>no pointcuts</code></td>
</tr>
</tbody>
</table>
# New Syntax and Semantics

## The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td><code>meta^2</code> aspect...</td>
<td><code>meta adviceexecution()</code>, <code>adviceexecution()</code></td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta</code> aspect...</td>
<td><code>adviceexecution()</code>, <code>meta &lt; pc &gt; and &lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 1</td>
<td>aspect...</td>
<td><code>&lt; pc &gt; except adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td>class...</td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>
### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td><code>meta^2</code> aspect...</td>
<td><code>meta adviceexecution()</code>, <code>adviceexecution()</code>, <code>meta^2 &lt; pc &gt;</code>,</td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta</code> aspect...</td>
<td><code>adviceexecution()</code>, <code>meta &lt; pc &gt;</code> and <code>&lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect</code>...</td>
<td><code>&lt; pc &gt;</code> except <code>adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class</code>...</td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>
The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td><code>meta^2 aspect...</code></td>
<td><code>meta adviceexecution()</code>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>adviceexecution()</code>, <code>meta^2 &lt; pc &gt;</code>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>meta &lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta aspect...</code></td>
<td><code>adviceexecution()</code>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>meta &lt; pc &gt; and &lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td><code>&lt; pc &gt; except adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td><code>no pointcuts</code></td>
</tr>
</tbody>
</table>
# New Syntax and Semantics

## The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3</td>
<td><code>meta^2 aspect...</code></td>
<td><code>meta adviceexecution()</code>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>adviceexecution()</code>, <code>meta^2 &lt; pc &gt;</code>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>meta &lt; pc &gt;</code> and <code>&lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 2</td>
<td><code>meta aspect...</code></td>
<td><code>adviceexecution()</code>,</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>meta &lt; pc &gt;</code> and <code>&lt; pc &gt;</code></td>
</tr>
<tr>
<td>Level 1</td>
<td><code>aspect...</code></td>
<td><code>&lt; pc &gt;</code> except <code>adviceexecution()</code></td>
</tr>
<tr>
<td>Level 0</td>
<td><code>class...</code></td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>
### The keyword `meta`

<table>
<thead>
<tr>
<th>Type Level</th>
<th>Type Definition</th>
<th>Allowed Pointcuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Level 3</td>
<td>$\text{meta}^2 \text{ aspect...}$</td>
<td>$\text{meta adviceexecution()}$, $\text{adviceexecution()}$, $\text{meta}^2 &lt; \text{pc}&gt;$, $\text{meta} &lt; \text{pc}&gt;$ and $\text{pc}&gt;$</td>
</tr>
<tr>
<td>Level 2</td>
<td>$\text{meta aspect...}$</td>
<td>$\text{adviceexecution()}$, $\text{meta} &lt; \text{pc}&gt;$ and $\text{pc}&gt;$</td>
</tr>
<tr>
<td>Level 1</td>
<td>$\text{aspect...}$</td>
<td>$&lt; \text{pc}&gt;$ except $\text{adviceexecution()}$</td>
</tr>
<tr>
<td>Level 0</td>
<td>$\text{class...}$</td>
<td>no pointcuts</td>
</tr>
</tbody>
</table>

Requirement:
Solution:

Requirement: Checking the LoD checker
Solution:

Requirement: Checking the LoD checker
Solution: meta aspect LoD
advises aspect LoD
Related Work - 1


Requirement: Checking the LoD checker
Solution: meta aspect LoD
advises aspect LoD

Testing Aspects [Sokenou, Hermann, 2005]

Requirement:
Solution:

Requirement: Checking the LoD checker
Solution: meta aspect LoD
advises aspect LoD

Testing Aspects [Sokenou, Hermann, 2005]

Requirement: "We need join points for advices to instrument aspects as well"
Solution:

<table>
<thead>
<tr>
<th>Requirement:</th>
<th>Checking the LoD checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution:</td>
<td>meta aspect LoD advises aspect LoD</td>
</tr>
</tbody>
</table>

## Testing Aspects [Sokenou, Hermann, 2005]

<table>
<thead>
<tr>
<th>Requirement:</th>
<th>&quot;We need join points for advices to instrument aspects as well&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution:</td>
<td>meta aspect AspectTester instruments aspect ObjectAdvisor</td>
</tr>
</tbody>
</table>
Conclusion and Outlook

Results

- Restoration of intuitive semantics (no more paradoxes!)
Conclusion and Outlook

Results

- Restoration of intuitive semantics (no more paradoxes!)
- Clear separation of aspects.
Conclusion and Outlook

Results

- Restoration of intuitive semantics (no more paradoxes!)
- Clear separation of aspects.
- Type system prevents programming errors.
Conclusion and Outlook

Results

- Restoration of intuitive semantics (no more paradoxes!)
- Clear separation of aspects.
- Type system prevents programming errors.
- (Slightly) more expressive pointcut language.
Conclusion and Outlook

Results

- Restoration of intuitive semantics (no more paradoxes!)
- Clear separation of aspects.
- Type system prevents programming errors.
- (Slightly) more expressive pointcut language.

Further Work

- Implementation of the type system (with abc group).
Results

- Restoration of intuitive semantics (no more paradoxes!)
- Clear separation of aspects.
- Type system prevents programming errors.
- (Slightly) more expressive pointcut language.

Further Work

- Implementation of the type system (with abc group).
- Systematic migration of current AOP applications.
Conclusion and Outlook

Results

- Restoration of intuitive semantics (no more paradoxes!)
- Clear separation of aspects.
- Type system prevents programming errors.
- (Slightly) more expressive pointcut language.

Further Work

- Implementation of the type system (with abc group).
- Systematic migration of current AOP applications.
- Collect empirical data on how many levels and/or whether intermediate levels are needed.
Conclusion and Outlook

Results

- Restoration of intuitive semantics (no more paradoxes!)
- Clear separation of aspects.
- Type system prevents programming errors.
- (Slightly) more expressive pointcut language.

Further Work

- Implementation of the type system (with abc group).
- Systematic migration of current AOP applications.
- Collect empirical data on how many levels and/or whether intermediate levels are needed.
- Generalization of the type system.