Symmetric Aspect-Oriented: Some Practical Consequences

Jaroslav Bálik    Valentino Vranič

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava

vranic@fiit.stuba.sk

NEMARA 2012 – AOSD 2012 – March 26, 2012, Potsdam, Germany
Symmetric aspect-oriented approaches promote aspect-oriented decomposition starting at the earliest phases of software development.

But academic symmetric aspect-oriented approaches seem to be too complicated for an average developer.

Can that be simplified to become widely accepted yet pertain essential features?

What out of that do we already have in industry?
Overview

1. Symmetry of Aspect-Oriented Approaches
2. Peer Use Cases
3. Feature Modeling
4. Aspect-Oriented Implementation in Established Programming Languages
5. Summary: Modularity Challenges and Innovations
Asymmetric and Symmetric AOP

- **Asymmetric AOP**: *aspects* (on one side) as something that affects the *base code* (on the other side)
  - Aspects are said to be woven into the base code
  - AspectJ and like—PARC\(^1\) AOP
  - Mainstream approach in AOP

- **Symmetric AOP**: aspects as partial *views* of classes
  - Functional classes are constructed by the compositions of selected *views*, i.e. aspects
  - Hyper/J—IBM Watson Research Center
  - No industry-strength languages

\(^1\)Palo Alto Research Center
A More Comprehensive View of Symmetry

- Here, symmetry is perceived mostly as element symmetry.
- A more comprehensive view of symmetry includes *join point symmetry* and *relationship symmetry*².

---

Peer Use Cases—Inherently Symmetrical
Feature Modeling

- Features are close to requirements
- Features are often given to developers as separate tasks
- If proper commit messages are used, the features can be tracked in version system
- Only temporary feature branches available
Developers apply symmetric aspect-oriented decomposition without actually being aware of it.

They are often forced to abandon this initial decomposition.

But some programming languages used in industry are close to symmetric aspect-orientation.
A trait is a unit that groups (related) methods unable to stand as full-fledged class.

Multiple traits can be composed with a single class.

An example in Scala:

```scala
class Student() {}

trait BasicStudent extends Student{
    var _name = ""
    var _surname = ""
    def setName(str:String) = { _name = str }
    def setSurname(str:String) = { _surname = str }
    def getName = _name
    def getSurname = _surname
}
```
trait PartTimeStudent extends Student {
    var tuitionFee = 0
    def payTuitionFee(amount: Int) = {
        tuitionFee = amount + tuitionFee
    }
    def tuitionFee = tuitionFee
}

object App {
    def main(args : Array[String]) {
        var student = new Student() with PartTimeStudent
            with BasicStudent
        ...
    }
}
Open Classes

- Ruby’s open classes enable to define parts of the same class at multiple places
- The concerns can be stored in different files
- The composition is made by importing the source files
- An example in Ruby:

```ruby
Student.rb:

class Student
  def initialize(name, surname)
    @name = name
    @surname = surname
  end
  def name; @name; end
  def surname; @surname; end
end
```
Open Classes (2)

PartTimeStudent.rb:

class Student
  def payTuitionFee(val)
    if @tuitionFee == nil
      @tuitionFee = val
    else
      @tuitionFee = @tuitionFee + val
    end
  end
  def tuitionFee @tuitionFee end
end

The composition—App.rb:

require "./Student.rb"
require "./PartTimeStudent.rb"
...

Prototype-Based Programming (1)

- Prototype-based programming is an object-oriented programming without classes.
- Prototype objects can be cloned and dynamically extended with new methods.
- The methods can be added in "batches" with each one representing another concern.
- An example in JavaScript:

```javascript
var student = {
    "_name": "",
    "_surname": "",
    setName:function(name) { this._name = name },
    getName:function() { return this._name },
    setSurname:function(surname) { this._surname = surname },
    getSurname:function() { return this._surname }
};
```
The `partTimeStudent` object is a clone of `student`:

```javascript
var Factory = function(){
}
Factory.prototype = student;
var partTimeStudent = new Factory();
```

Methods and attributes necessary for the role of a part-time student are added to it:

```javascript
partTimeStudent['_tuitionFee'] = 0;
partTimeStudent['payTuitionFee'] =
    function(val) { this._tuitionFee = this._tuitionFee + val };
partTimeStudent['getTuitionFee'] =
    function() { return this._tuitionFee };
```
Symmetric aspect-oriented programming can be emulated to some extent in asymmetric approaches

- Keep the base as thin as possible and build everything with aspects
- Inter-type declarations establish the structure, including initial method bodies
- The behavior is then implemented by advices
An example in AspectJ

```java
public class Student { }

public aspect BasicStudent {
    private String Student.name = null;
    private String Student.surname = null;
    public Student.new(String name, String surname) {...}
    public String Student.getName() {...}
    public String Student.getSurname() {...}
}

public aspect PartTimeStudent {
    private double Student.tuitionFee = 0;
    public void Student.payTuitionFee(double tuitionFee) {...}
    public double Student.getTuitionFee() {...}
}
```
The Key Modularity Challenges That Remain Unaddressed

- The design gap: no design notation used in industry enables aspect-oriented modeling
- Identify further features in industry-strength languages close to symmetric AOP
What Key Innovations May Help Address the Modularity Challenges?

- Constructs of existing industry-strength programming languages in which aspect-oriented programming is possible should be improved to provide better symmetric aspect orientation.
- To spread the knowledge about symmetric aspect-oriented development to the industry.
- Comprehensive studies and real applications of symmetric aspect-oriented development are needed.