Composition and Categorization of Aspect-Oriented Design Patterns

Radoslav Menkyna[1] Valentino Vranić[2] Ivan Polášek[2]

Softec, spol. s.r.o.
Kutuzovova 23, 83103 Bratislava 3, Slovakia
radoslav.menkyna@softec.sk

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology,
Ilkovičova 3, 84216 Bratislava 4, Slovakia
vranic@fiit.stuba.sk

SAMI 2010, January 28–30, 2010, Herľany, Slovakia

Overview

Patterns and Aspects

Structure of Aspect-Oriented Design Patterns

Pattern Composition

Patterns

Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution.
–Alexander, The Timeless Way of Building

- According to Alexander's original idea, patterns are indivisible of the pattern language
- Software patterns: design, analysis, architectural, organizational...
- Some pattern languages are available, but software patterns are applied mostly individually
- Pattern composition: a subsequent interrelated application of two patterns to a problem at hand

Aspects

- Advanced software decomposition and composition approach: decomposition into multiple views developed separately and composed as needed
- Commonly denoted as aspect-orientation
- PARC AOP and AspectJ prevail: untangling crosscutting concerns
- But there are quite different yet still aspect-oriented approaches

Aspect-Oriented Design Patterns

- Aspect-oriented design patterns are being identified mainly within AspectJ
- The question remains whether they are general enough or are they merely AspectJ idioms (set aside here)
- Aspect-oriented programming is known for obliviousness of the affected concerns
- *How oblivious are already applied patterns to addition of other patterns?*

Aspect

- The main construct in PARC aspect-oriented programming is an aspect
- Main parts of an aspect:
  - Pointcuts: specifying the join points the aspect affects
  - Advices: implementing the affecting functionality
  - Inter-type declarations: introducing new fields and methods, inheritance relationship, warnings, compile errors, softened exceptions, and annotations into types

Example: Cuckoo's Egg

- Put another object instead of the one that the creator expected to receive

```
public aspect MyClassSwapper {
    public pointcut myConstructors():
        call(MyClass1.new()) || call(MyClass2.new());

    Object around(): myConstructors() {
        return new AnotherClass();
    }
}
```

Aspect-Oriented Design Pattern Categories

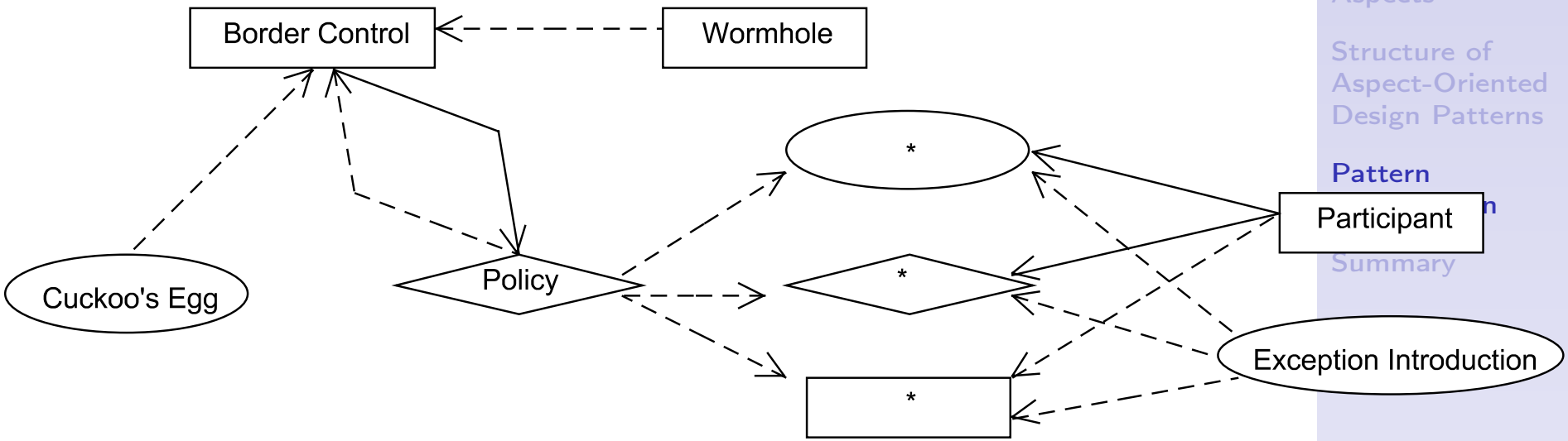
- Each aspect-oriented design pattern comprises at least one aspect
- One of the three main parts of an aspect prevails in achieving the purpose of the pattern
- According to the prevailing part, aspect-oriented design patterns can be divided into three categories:
  - Pointcut patterns: Border Control, Wormhole, and Participant
  - Advice patterns: Cuckoo's Egg and Worker Object Creation
  - Inter-type declaration patterns: Policy and Default Interface Implementation

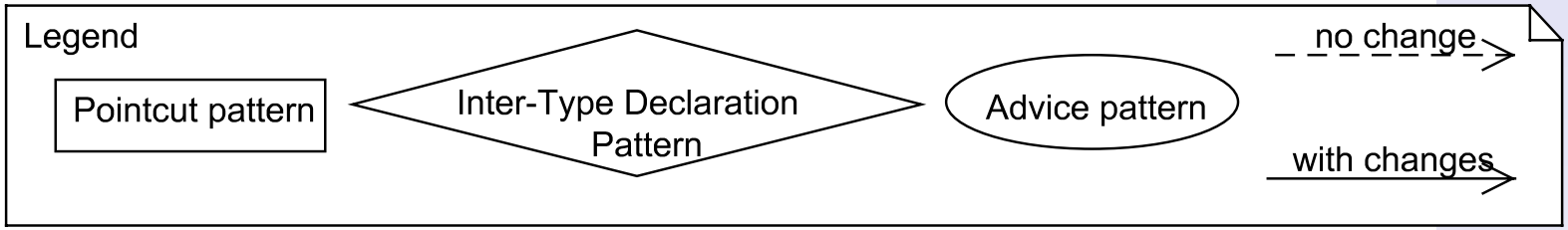
A Study

- Overcoming the class deprecation problem in team development
- Subsequent application of four patterns:
  1. Policy
  2. Border Control applied to Policy
  3. Cuckoo's Egg applied to Border Control
  4. Exception Introduction applied to Cuckoo's Egg and Border Control
- The class deprecation study served as a starting point
- Pattern composition in reverse order has been considered, too
- The composition of other patterns has been analyzed

Regularity in Aspect-Oriented Design Pattern Composition

- The composition of aspect-oriented design patterns is substantially affected by their structural category





Composing Policy with Border Control (1)

- The warning of deprecated class use implemented as a Policy pattern:

```
public aspect Warning {
    declare warning: call(*.OldClass.new()):
        "Class OldClass deprecated.";
}
```

- A Border Control pattern to allow the use of OldClass within the testing package and third party code:

```
public aspect Regions {
    public pointcut Testing():
        within(com.myapplication.testing.+);
    public pointcut MyApplication():
        within(com.myapplication.+);
    public pointcut ThirdParty():
        within(com.myapplication.thirdpartylibrary.+);
    public pointcut ClassSwitcher():
        within(com.myapplication.ClassSwitcher);
}
```

Composing Policy with Border Control (2)

- The original Policy pattern instance (repeated):

```
public aspect Warning {
    declare warning: call(*.OldClass.new()):
        "Class OldClass deprecated.";
}
```

- Necessary modifications of the Policy pattern:

```
public aspect Warning {
    protected pointcut allowedUse():
        Regions.ThirdParty() || Regions.Testing();

    declare warning: call(Display.new()) && !allowedUse():
        "Class OldClass deprecated.";
}
```

Summary

- Proposed a categorization of aspect-oriented design patterns according to their structure
- Study of the composition of aspect-oriented design patterns of different categories with respect to the stability of the already applied patterns
- Further work
  - Explore the possibilities of employing aspect-oriented design patterns and their compositions in capturing changes in a pluggable and reapplicable way
  - Support for instantiation of aspect-oriented patterns
  - Seek further parallels between categorization of GoF patterns and our categorization of aspect-oriented patterns