Towards Combining Aspect-Oriented Design Patterns

Radoslav MENKYNA*

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia
xmenkyna@is.stuba.sk

Abstract. Although aspect-oriented paradigm is quite new, patterns are emerging in it already. This article provides an overview of common aspect-oriented design patterns. The article discusses how these patterns can be combined. Certain dependence between the design pattern structure and its combination ability has been identified. Knowing a structure of patterns, it is possible to say whether adding a new pattern to existing one can be made with or without change of existing pattern. Classification according to structure into pointcut design patterns and advice design patterns is proposed.

1 Introduction

With increasing size of the project grows also the complexity of problems that have to be solved. It is natural to use simple smaller building blocks to produce bigger structures. That's why it is natural to combine design patterns to solve complex problems.

This article identifies some regularities in combining aspect-oriented design patterns according to their classification. Design patterns are classified according to structure into pointcut and advice design patterns. Certain dependence between the combination ability of specific pattern and its structure has been identified. According to structure of the pattern it is possible to say whether adding some pattern to existing pattern can be made with or without changes of existing pattern. The combination of design patterns allows using their features together, but also it can lead to new features.

The rest of the article is organized as follows. In Section 2 an overview of aspect-oriented design patterns can be found Section 3 presents a classification of aspect-oriented

* Supervisor: Dr. Valentino Vranić, Faculty of Informatics and Information Technologies STU in Bratislava.

IIT.SRC 2007, Bratislava, April 18, 2007, pp. 1–8.

design patterns according to Structure. Section 4 discusses how design patterns can be combined and presents the dependence between structure and combination ability of a pattern. Section 5 provides an overview of related work Section 6 represents the conclusion and future work.

2 Overview of Aspect-Oriented Design Patterns

Despite aspect-oriented paradigm is just spreading, some strategies and idioms have already been substantially generalized, do not rely on a particular language any more, and as such can be accepted as design patterns. This section will present an overview of common aspect-oriented design patterns.

2.1 Wormhole

The Wormhole pattern [4] connects the callee with caller in such a way that they share their context information. It creates a direct connection between two levels in the call stack. This is very helpful when additional context information has to be added [4]. Instead of adding extra parameters in each method in the control flow or using a global storage, this pattern can be used.

2.2 Exception Introduction

In some cases when the aspect is used to implement some crosscutting concerns checked exceptions have to be caught in its advice. This usually happens when methods of Java libraries, which declare to throw such exceptions, are used in its advice. In AspectJ,[1] advice cannot declare to throw a checked exception unless the advised joint point declared this exception. The base concern logic cannot declare those exceptions because it simply does not know anything about the logic used in the crosscutting concern [4].

The Exception Introduction pattern [4] suggest that the checked exceptions should be caught and simply wrapped into new concern-specific runtime exceptions. Such exceptions can be then thrown to higher level, where they can be unwrapped and the real cause of exception revealed.

2.3 Participant

Usually, aspects try to introduce some behavior to a base concern in such a way that the base concern is not aware of the aspect. In this pattern, the roles swap: an aspect makes classes to participate. This is needed for the purpose the Participant [4] is trying to achieve. In some cases, defining pointcuts only by the means of language syntax is not sufficient.

[1] This problem is common to many aspect-oriented languages.

For example, if the advice should affect only methods with certain characteristics, one cannot decide only according to their names whether to include them in the pointcut or not. Only the creator of these methods knows their characteristics. This is where comes from the idea that classes themselves should express if they want to be advised. If they want to participate they simply define an appropriate pointcut in them [4].

2.4 Cuckoo's Egg

Cuckoo's Egg design pattern [5] is quite simple but powerful. It expresses how powerful aspect-oriented programming can be. It is used to control or change the objects created by the constructor call. This means that with this pattern it is possible to change the type of the object being instantiated [5].

2.5 Director

The Director pattern [5] can be used to define some roles or behavior to an unknown number of classes. A role can be defined without knowing the particular class it will be applied to. A pattern can be used to define some logic in an abstract aspect without knowing the classes this logic will be applied to. The Director can also implement some relationships within abstract entities [5].

2.6 Border Control

The Border Control design pattern [5] is used to define some reasonable regions in the application. These regions are later reused by other aspects to ensure they are used only in correct scope. Use of this pattern is also convenient when the changes in structure of the application are expected. After such changes only declarations of regions in border control aspect are changed and other aspects which are reusing these declarations will be also affected [5].

2.7 Policy

The main idea of Policy pattern is to define some policy or rules within the application. The rules can wary from suggestions and warnings to overriding methods, classes or libraries. This is very useful in some project where many developers are involved [5].

2.8 Worker Object Creation

Worker Object Creation pattern [4] has a widespread use. For example, it may be used when the use of the proceed call in an object-oriented context is needed or when the proceed call should be executed in a different thread. This can be used with Java Swing Framework, where all calls which update the GUI must be performed inside the event dispatch thread [6]. Another example of the situation when the proceed call should be

executed in a different thread is improving responsiveness of GUI applications which perform complex computations (e.g., authorization and transaction management) [4].

This pattern can also be used to advise the proceed call. This is desired when the aspect contains an around advice and the algorithm in the advice itself should be, for example, traced or logged [6].

3 Aspect-Oriented Design Pattern Classification According to Structure

Interesting criterion for classification of aspect-oriented design patterns could be the structure of an aspect-oriented design pattern.

Each design pattern is usually represented by one or more aspects. Structurally, an aspect consists of three main components: inter-type declarations, pointcuts and advices. Generally, the structure of an aspect that represents a design pattern can be divided into two categories. These categories differ according to which component is more significant for the main meaning for the purpose of the design pattern or, in other words, which component is crucial to understand or achieve the logic of the design pattern.

If main purpose is realized by pointcuts, advice is usually not presented or is very simple. For example, an advice only presents how pointcuts would be used in it, but the advice logic is not represented. In the category where advices have the main role, pointcuts are usually declared as abstract or they are only symbolic.

Thus, according to their structure, aspect-oriented design patterns can be divided into two categories: pointcut design patterns and advice design patterns. Patterns from listed in the overview were divided according to structure into these categories:

- **Pointcut design patterns**: Wormhole pattern, Participant pattern, Border Control pattern

- **Advice design patterns**: Worker Object Creation pattern, Exception Introduction pattern, Cuckoo's Egg pattern, Director pattern, Policy pattern.

In most cases patterns were easy to classify by this criteria. It is possible to point out the Participant pattern (Section 2.3) where the pointcuts crucial to logic of the design pattern are not defined the in aspect but in plain Java classes. The Director design pattern (Section 2.5) has a logic that can be sometimes expressed in a more complex way than just by an advice (e.g., using interfaces or additional method definitions). In wormhole pattern (Section 2.1) the advice is present only to show how to use pointcuts in it. At last, the Policy design pattern (Section 2.7) is in this paper treated like the advice pattern, although it could be a representant of new inter-type declaration design patterns category. From the point of combination these two categories have the same behaviour.

4 Combination of Aspect-Oriented Design Patterns

Aspect-oriented design patterns are defined in a modular way. They can be usually implemented by one or more aspects, but in most cases their logic is concentrated in one place. This comes from the nature of aspect-oriented programming. Aspects usually alter behavior of base concerns without requiring awareness on their side. This makes the combination of aspect-oriented design patterns easier than the combination of object-oriented design patterns is. In general, a design pattern can be combined with many other design patterns. The question is whether this combination would be useful and meaningful. Due to this, we may expect that aspect-oriented design patterns more easily form pattern languages.

In the following sections a dependence between the structure and combination ability of patterns together with examples of combinations will be presented.

4.1 Dependence Between Structure and Combination Ability of Patterns

There is a connection between the structure of aspect-oriented design patterns and the way how they can be combined with other aspect-oriented design patterns. The combination of aspect-oriented design patterns is substantially affected by their structural type, this means is possible to make statements about the way how this pattern could be combined with other patterns.

It seems that a combining of a pointcut design pattern (Section 3) with another pointcut design pattern does not require changes of existing design pattern. In this kind of combination, the new pattern will reuse the pointcuts of the existing pattern. Adding pointcut a design pattern to an advice design pattern usually requires changes in the advice design pattern.

On the other hand, a combination of an advice design pattern with another design pattern of any structural type can be done without changes to existing design pattern. Examples of such combinations can be seen in sections: 4.2, 4.3. Dependence between the structure of design patterns and the way how they can be combined is summarized in Table 4.1.

Tab. 1. Dependence between the structure of design patterns and the way how they can be combined.

	pointcut design pattern	advice design pattern
pointcut design pattern	without change	change required
advice design pattern	without change	without change

4.2 Adding a Feature

Sometimes, a design pattern only adds some feature to the system. If this feature is needed by another design pattern, the patterns can be used together. Aspect-oriented design patterns are usually represented by one or more aspects. From the nature of aspects some design pattern can be added to another without the modification of the existing pattern.

An example of such a pattern that can be combined with almost any other pattern is the Exception Introduction design pattern (Section 2.2). This pattern adds the ability to use exceptions in advices in a proper way. When this feature is needed in another design pattern, the Exception Introduction pattern can simply be added to the program without having to make any change in existing patterns. Also Policy design pattern (Section 2.7) can be used together with another design patterns without any changes to those patterns.

Example of exception introduction pattern adapted from [4]:

```
public abstract aspect ExceptionIntroductionAspect {
        abstract pointcut operations();
        // pointcut operations defines where should exception occur.
        // When this is defined as another design pattern exception introduction is used together with an existing pattern
        // and any change of its code is required.
        Object around() : operations() {
                try {
                return proceed();
        } catch (CheckedException ex) {
                throw new RuntimeException(ex);
                // CheckedException will be caught and new runtime exception will be thrown.
                }
        }
}
```

Adding pointcut design pattern to an advice design pattern requires usually a change in existing advice pattern. Example of such a pattern is the Border Control design pattern (Section 2.6). This pattern defines regions that are later used by other aspects or design patterns. This suggests that after adding a feature represented by this pattern existing aspects and patterns have to be altered.

Assume the pointcuts of a particular pattern in a growing application can no longer be defined in a simple way because it is not certain whether the pattern should be applied to new classes. Such a pattern can be combined with a Participant pattern (Section 2.3) and the class can participate in the application of this pattern. Due to the implementation of the Participant pattern, the existing pattern code must be altered.

All the combinations in this section are summarized at Figure. 1. There are three groups of design patterns on the figure. Any pattern from the groups on a sides can be combined with any pattern in the middle. Combining pattern from the left group, advice design pattern, can be usually done without change. When using pattern from the right group, pointcut design pattern, changes of existing design pattern will be required. All patterns in the middle group are advice design patterns.

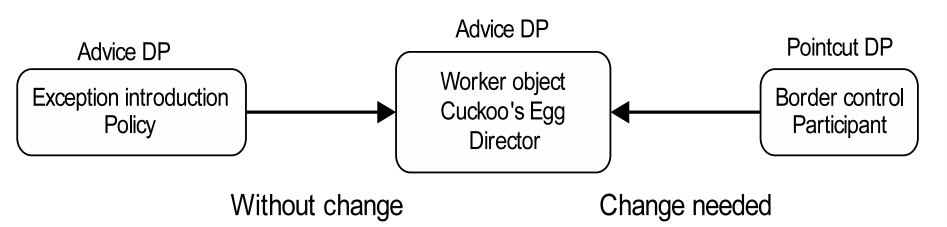

Fig. 1. Illustration of possible combinations of aspect-oriented design patterns.

4.3 Achieving New Functionality

Aspect-oriented design patterns can also be combined to achieve new functionality. A simple example of such a use is the combination of the Policy design pattern (Section 2.7) with Cuckoo's Egg design pattern (Section 2.4). The Policy pattern in most cases only defines some warnings or suggestions upon breaking the specific policy. By combining it with the Cuckoo's Egg design pattern it is possible to go further by not only showing warning, but also override the use of specific classes and their methods.

5 Related Work

In GoF description of object-oriented design patterns [1] a section called related patterns can be found. This section lists the names of the patterns often used with the pattern. This in other words suggests which patterns can be combined with the pattern to solve some problem. GoF also presented graphical representation of these relations. From this representation, groups of patterns that are often combined can bee seen.

Object-orientated design patterns were reimplemented in AspectJ [3]. In many cases better modularity was achieved, so the design pattern structure became clearer and simpler to understand [3]. These patterns could be also combined with the intrinsic aspect-oriented design patterns and better modularity achieved by reimplementation would ease this combination.

Combination of idioms is similar to combination of design patterns. Idioms can be generalized to become design patterns. Also idioms can be combined in various ways to achieve a solution to some problem [2].

6 Conclusion and Future Work

The Article proposed ways how the aspect oriented design patterns can be combined. Certain dependence between the design pattern structure and its combination ability was discussed. The combination of the pointcut design pattern with another pointcut pattern is usually done without need to modify the existing pattern. The combination of the pointcut design pattern with an advice design pattern requires a change of existing

structures during the implementation. Adding an advice design pattern to another design pattern leads usually to the implementation where no change of existing pattern is needed.

This paper provided an overview of common aspect-oriented design patterns. An alternative way of classification according to the structure of an aspect that represents the design pattern was presented. Aspect-oriented patterns were divided into two categories: pointcut design patterns and advice design patterns.

As a future work I would like to examine some inter-type declaration design patterns which can form another structural category. More combinations of the aspect-oriented design patterns should be examined. It is expected that new patterns will arise, which will bring new possibilities into the combinations. There are also many object-oriented design patterns reimplemented in AspectJ [3]. Combination of these patterns with intrinsic aspect-oriented design patterns could be also interesting.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG1/3102/06.

References

[1] Gamma, E., et al.: *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] Hanenberg, S., Schmidmeier, A., Unland, R.: AspectJ Idioms for Aspect-Oriented Software Construction. In: *Proceedings of EuroPLoP 2003*, 2003.

[3] Hannemann, J., Kiczales, G.: Design pattern implementation in Java and AspectJ. In: *Proceedings of the 17th conference on Object-oriented programming, systems, languages, and applications (OOPSLA)*, 2002, pp. 161–173.

[4] Laddad, R.: *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning, 2003.

[5] Miles, R.: *AspectJ Cookbook*. O'Reilly, 2004.

[6] Schmidmeier, A.: Patterns and an Antiidiom for Aspect Oriented Programming. In: *Proceedings of 9th European Conference on Pattern Languages of Programs, EuroPLoP 2004*, Irsee, Germany, 2004.