Pattern Transformation for Two-Dimensional Separation of Concerns

Xiaoqing Wu
Department of Computer and Information Sciences
The University of Alabama at Birmingham
Birmingham, AL 35294-1170, USA
wuxi@cis.uab.edu

ABSTRACT
Design patterns are applied in software development to decouple individual concerns, so that a change in a design decision is isolated to one location of the code base. However, multi-dimensional concerns exist in software development and some concerns are even mutually exclusive. Therefore, no single design pattern offers a panacea toward addressing problems of change evolution. By analyzing the matrix of concerns during the software development process, this dissertation abstract describes a paradigm for two-dimensional separation of concerns based on pattern transformation. In particular, it shows an example to transform code back and forth between an object-oriented implementation of the Inheritance pattern and an aspect-oriented implementation of the Visitor pattern. The approach allows the same software to be evolved along different dimensions, enabling developers to choose the most appropriate dimension for a given task.

Categories and Subject Descriptors

Keywords
Design Patterns, AOP, Separation of Concerns, Pattern Transformation.

1. INTRODUCTION
One general intention of design patterns is to decouple individual concerns, so that a change in a design decision is isolated to one location of the code base. Each design pattern is designed to facilitate one kind of change, i.e. changes in one dimension. However, software evolution can happen in multiple dimensions [1] and each dimension has its own best-fit modularization requirements, which may exclude requirements of other patterns. Therefore, none of the design patterns is a panacea to fulfill the multi-dimensional evolution needed during software development.

Aspect-Oriented Programming (AOP) provides special language constructs that modularize concerns that crosscut conventional program structures (e.g., class hierarchies of object-oriented programs). This offers a second dimension for software modularization besides object-orientation. Recent observations have indicated that some object-oriented design patterns (e.g., Visitor, Mediator and Abstract Factory) have basic AOP characteristics: without them the structure and behavior characteristics are scattered throughout the code base. Applicability of AOP toward modularizing object-oriented design patterns has been heavily researched [2]. One example used in this paper is the aspect-oriented Visitor pattern introduced in [3].

This extended abstract describes dissertation research that explores two-dimensional separation of concerns in software development in the form of a concern matrix. It demonstrates the interchangeability between the object-oriented Inheritance pattern [4] and the aspect-oriented implementation of Gang-of-Four (GoF) patterns [5] based on pluggable aspects. The pattern transformation approach combines object-oriented design pattern principles with aspect orientation and leads to a two-dimensional approach toward software evolution, which enables developers to choose the most appropriate dimension for a given task.

2. DESIGN PATTERNS
In this section, the usage and limitations of the Inheritance and Visitor patterns are briefly explored as applied to a simple payroll system. More details regarding the description and implementation of each design pattern can be found in [6].

There are three kinds of employees in the system: regulars, executives and contractors. A straightforward way to build the system in an object-oriented fashion is to apply the Inheritance pattern [4] to create a super class named Employee with abstract/concrete methods such as wage (calculate the salary) and print (output the basic information of an employee). Afterwards, a subclass for each kind of employee can be defined to implement and extend Employee. The aspect-oriented Visitor pattern differs from the Inheritance pattern in that it encapsulates and relocates all the methods pertaining to one functional operation into a single visitor aspect. For example, the listing below is a sample aspect for wage calculation using AspectJ [7]. As can be seen, the operation methods of various subject classes are declared as inter-type declarations inside the aspect. In the implementation of the payroll system, it is clear that these two patterns both have their benefits and limitations. To put it briefly, the Inheritance pattern assists in flexibly adding new types of employees, but is unsuitable for adding new functionality (e.g., tax calculation) to the system; the Visitor pattern is useful for adding operations, but inappropriate for adding new employee types.

```java
1. aspect Wage {
2.   public abstract double Employee.wage();
3.   public double Executive.wage() {
4.     wage = baseWage + bonus;
5.     return wage;
6.   }
7.   public double Regular.wage() {...
8.   public double Contractor.wage() {...
9. }
```
3. CONCERN MATRIX AND PATTERN TRANSFORMATION

Reflected by the above exploration, the abstraction of all the necessary constructs in the system can be considered as a two-dimensional (2D) subject-function concern matrix (please refer to the pictures in the associated poster). Each column of the matrix can represent an employee type, and each row represents the same functionality on all kinds of employees. Every employee has several operations and every operation crosscuts multiple employee types. From an orientation point of view, one column represents a class and one row represents an aspect. If all of the artifacts are modularized vertically, an instance of the Inheritance pattern emerges, which could be realized using object-orientation. Correspondingly, if the matrix artifacts are modularized horizontally, an instance of the Visitor pattern emerges, which can be implemented utilizing aspect-orientation. Therefore, the 2D subject-function concern matrix is essentially a 2D class-aspect matrix.

The 2D concern matrix reveals that an ideal solution should provide two-dimensional separation of concerns [1] and a facility to make the two dimensions transferable. As a result, pattern transformation based software development approach is developed, in which the Inheritance pattern is implemented using pure Java and the Visitor pattern is implemented using Java and AspectJ. These two patterns are transferable in the development process and only one pattern exists at a time. Because the implementations of the two patterns have the same set of operations and both use Java syntax in method implementation, the transformation between two patterns can be achieved simply by relocation of all the methods, i.e. from AspectJ aspects to Java classes (aspect weaving) or from Java classes to AspectJ aspects (aspect unweaving). The whole software development paradigm is outlined by the following steps. The aspect weaving and unweaving strategy is described in [3].

1. Initially use either the Inheritance pattern or the Visitor pattern to build the system.
2. Once new functional behaviors need to be added or old functions need to be changed, transform the Inheritance pattern to the Visitor pattern (if it is not) by unweaving the operation methods of each class into individual aspect specifications, and then change the operations inside a visitor aspect or add new visitors to the system.
3. Once new subject classes need to be added, transform the Visitor pattern to the Inheritance pattern (if it is not) by weaving the operations in every aspect into the corresponding class, and add the new subject classes using the Inheritance pattern.

4. EVALUATION AND EXTENSION

The examination and extension of the research has been done in two directions. First, in addition to the sample payroll system introduced in this paper, the Inheritance pattern and Visitor pattern transformation scheme has also been utilized successfully in the field of compiler design [3]. Another extension is to generalize the idea of pattern transformation on other design pattern pairs such as Inheritance vs. Abstract Factory, Inheritance vs. Mediator and Inheritance vs. Observer. Due to the same reason as seen in the Visitor pattern, the Abstract Factory, Observer, and Mediator patterns all have drawbacks in adding new kinds of subject classes. Based on the similar 2D class-aspect concern matrix, the problem can be eliminated by transforming those patterns back and forth to the Inheritance pattern to accommodate change decisions, as is detailed in [6].

5. RELATED WORK AND CONCLUSION

Tarr et al. first introduced the concept of Multi-Dimensional Separation of Concerns (MDSOC) [1], which is implemented using hyperspaces that allow developers to identify concerns and dimensions, and align units according to concerns. Our contribution differs from the above in that we simplify the complexity of MDSOC by only focusing on the two orthogonal dimensions and use a straightforward aspect weaving and unweaving approach to modularize different dimensions of concerns.

There are always multi-dimensional concerns in software development. No single design principle or pattern offers a panacea toward addressing problems of change evolution. Transformation techniques applied to design patterns offer an alternative to alleviating this problem. This abstract analyzed the essence of the two-dimensional concern matrix and presented a pattern transformation approach for software evolution in two dimensions using object-orientation and aspect-orientation. Due to space restrictions, several implementation details are omitted in this paper. Interested readers may refer to the project website at the project web site (http://www.cis.uab.edu/softcom/cde) for more implementation details.

6. REFERENCES