International Journal of Software Innovation
Volume 8 • Issue 1 • January-March 2020

Visitor Design Pattern Using Reflection Mechanism

Bilal Hussein, Lebanese University, Beirut, Lebanon
Aref Mehanna, Lebanese University, Beirut, Lebanon
Yahia Rabih, Lebanese University, Beirut, Lebanon

ABSTRACT

Design patterns of today play a fundamental role in software development and implementation and provide a wide range of design solutions for recurring problems. Most research in this area focus on the creation and update of design patterns in order to fill all the gaps produced by their original structures. The purpose of this article is to present the visitor design pattern, to show its advantages in the software development process, and to provide it in a new version that allows the software to be easily upgraded without making complex modifications. This contribution consists in updating the structure and implementation of the Visitor design pattern using the reflection mechanism.

KEYWORDS

Design Pattern, Introspection, Reflection, Software Development Process, Software Evolution, Software Maintenance, Visitor Pattern

INTRODUCTION

Design patterns play an essential role in Software Development Process (SDP) and are widely accepted as useful concepts for documenting, guiding and evolution of software systems (Le Guennec et al., 2000; Hsueh et al., 2011). One of the main advantages of design patterns in SDP is to improve the programmer's productivity and software quality (Prechelt et al., 2002; Khomh & Guéhéneuc, 2018). Moreover, design patterns provide a wide range of design solutions for recurring problems (Gamma et al., 1995). In recent decades, these solutions are taken into consideration in the SDP process and started using the new concepts of object-oriented design (OOD). These concepts are based on classes, interfaces, methods, and fields. The majority of researches in the area of design patterns centers on the creation and update of design patterns in turn to fill all the gaps produced by their original structures. Original design pattern structures are described by class diagrams based on Object Modeling Technique (OMT) (Rumbaugh et al., 1991). Recently, the unified modeling language (UML) provides better support for design patterns (Sunyé et al., 2000; France et al., 2004; Mak et al., 2004; Dennis et al. 2015). In this paper, we use the UML class diagram because it is a de facto standard and it is well known among developers and Java as Object Oriented Language (OOL) for implementation. The purpose of this paper is to present the Visitor Design Pattern (VDP) as one of the 23 patterns invented

DOI: 10.4018/IJSI.2020010106

by GoF (Gamma et al., 1995), to show its advantages in SDP and to provide a new innovative version of it that allows software to be easily upgraded without making modifications on its original classes.

The report issued by Khashayar shows that the VDP has a good impact on software quality characteristics and especially on the expendability, simplicity, generality, software independence, learnability and scalability (Khashayar & Yann-GaÄel, 2004). A study conducted by Nanthaamornphong and Wetprasit shows that VDP improves the software design simplicity (Nanthaamornphong & Wetprasit, 2014). Moreover, the VDP has a good impact on software maintainability (Jeanmart et al., 2009). April & Abran summarize four categories of software maintenance: corrective, perfective, adaptive and preventive. Adaptive and preventive maintenance concern the ability of a system to answer a request for software's improvement (e.g., adding a new task or responsibility) (April & Abran, 2006). VDP lets you define a new operation (method) without changing the classes of the elements on which it operates (Gamma et al., 1995). The principle is to create an interface Visitor that contains an abstract method for each visited class.

For instance, if we have two visited classes A and B, the Visitor interface will be as follow:

```
public interface Visitor {
<T> T visit(A a);
<T> T visit(B a);
}
```

Moreover, we have to add, in each visited class (maintainable class), a generic method called 'accept', and prepare it for accepting visitor objects that define new responsibilities (methods). The body of the method 'accept' in each visited class is:

```
public class A {
public <T> T accept(Visitor v) {
return (T)v.visit(this) ;
}
}
public class B {
public <T> T accept(Visitor v) {
return (T)v.visit(this) ;
}
```

The concrete visitor class which implements each visit() method is as follow:

```
public class concreteVisitor implements Visitor {
public <T> T visit(A a){
.........
}
public <T> T visit(B a){
.........
}
}
```

This original visitor design pattern has limitations. Patti and Hill presented in their survey report a summary of 10 limitations: Prior knowledge of the arguments and return type, adding new elements is difficult, necessity of accept method, partial visitation, structure extension requires regeneration of traversal code, little traversal control, implementation inheritance not supported, violation of