Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 14 Issue 6 Version 1.0 Year 2014
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Strategy Design Pattern

By Mrs. Renu Bala & Mr. Kapil Kumar Kaswan

CDLU SIRSA, India

Abstract- Design patterns usually describe abstract systems of interaction between classes, objects, and communication flows. So, a description of a set of interacting classes that provide a generalized solution framework to a generalized/specific design problem in a specific context can be said as a design pattern. There are many design patterns that can be used to solve real-life problems, but it remains very difficult to design, implement and reuse software for complex applications. Examples of these include enterprise system, real-time market data monitoring and analysis system. Design patterns provide an efficient way to create more flexible, elegant and ultimately reusable object-oriented software. Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice". The solutions of the given problems are expressed in terms of objects and interfaces. Among 23 design patterns, Strategy pattern defines an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a Concrete Strategy. In accounting framework one thing is mostly needed that is tax calculation. To solve this problem author in the current study has chosen the strategy pattern.

Keywords: *design pattern, context, strategy, object, concretestrategy.*

GJCST-C Classification : *I.5*

STRATEGYDESIGNPATTERN

Strictly as per the compliance and regulations of:

Strategy Design Pattern

Mrs. Renu Bala [α] & Mr. Kapil Kumar Kaswan [σ]

Abstract- Design patterns usually describe abstract systems of interaction between classes, objects, and communication flows. So, a description of a set of interacting classes that provide a generalized solution framework to a generalized/specific design problem in a specific context can be said as a design pattern. There are many design patterns that can be used to solve real-life problems, but it remains very difficult to design, implement and reuse software for complex applications. Examples of these include enterprise system, real-time market data monitoring and analysis system. Design patterns provide an efficient way to create more flexible, elegant and ultimately reusable object-oriented software. Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice". The solutions of the given problems are expressed in terms of objects and interfaces. Among 23 design patterns, Strategy pattern defines an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a Concrete Strategy. In accounting framework one thing is mostly needed that is tax calculation. To solve this problem author in the current study has chosen the strategy pattern.

Keywords: design pattern, context, strategy, object, concretestrategy.

I. Introduction

A design pattern is a generic solution that has been observed in multiple instances to help resolve a particular problem within a known context. Design patterns provide an efficient way to create more flexible, elegant and ultimately reusable object-oriented software. Design methods are supposed to promote good design, to teach new designers how to design well and to standardize the way designs are developed. Typically, a design method comprises a set of syntactic notations usually graphical and a set of rules that govern how and when to use each notation. It will also describe problems that occur in a design, how to fix them, and how to evaluate a design. Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice [1]. The solutions of the given problems are expressed in terms of objects and interfaces. Design patterns are being increasingly used in software design. Design patterns are a good means for recording design experience as they systematically name, explain and evaluate important and recurrent designs in software systems. They describe problems that occur repeatedly, and describe the core of the solution to that problem, in such a way that this solution can be used many times in different contexts and applications. A good design is a good solution regardless of the technology. And no matter how good the technology may be, it is only as good as its design, and specifically the implementation of that design. In fact, a great design with older technology may still be good, but a bad design with new technology is usually just bad. A design pattern is a form of design information and the design that worked well in past will be used in future in any application similar to existing application which uses these designs. These design information can help both the experienced and the novice designer to recognize situations in which these designs can be reused. There are three categories of design patterns: Creational, structural and Behavioral.

Author α σ: Department of Computer Science and Application Chaudhary Devi Lal University Sirsa, Haryana, India.
e-mails: renubala.gghsodhan@gmail.com,

II. Net Framework

A .net is a new software platform for the desktop and the Web. The .NET Framework is an integral Windows component that supports building and running the next generation of applications. The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework [2]. The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented program - ming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.

- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code [2].

III. Strategy Pattern

Strategy - defines an interface common to all supported algorithms. Context uses this interface to call the algor - ithm defined by a ConcreteStrategy.

ConcreteStrategy - each concrete strategy implements an algorithm.

Context

- contains a reference to a strategy object.
- may define an interface that lets strategy accessing its data.

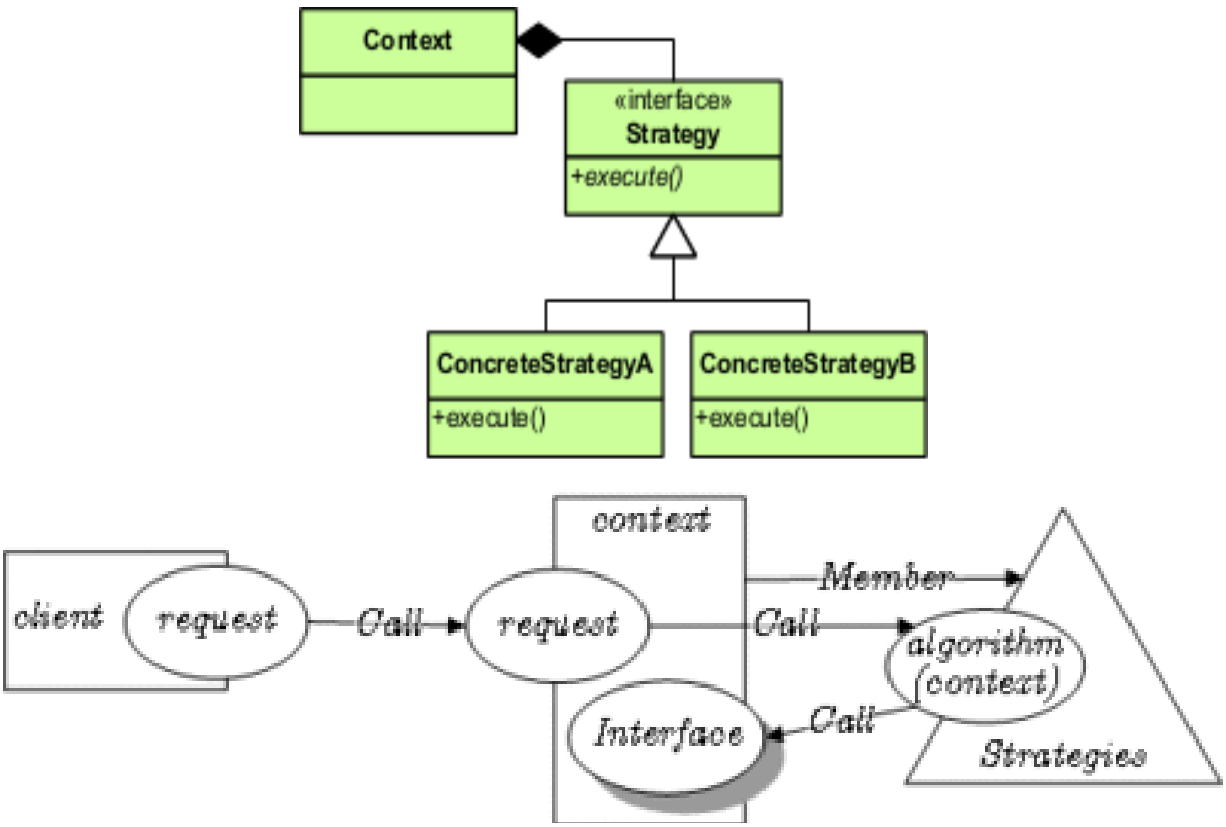
The Context objects contains a reference to the ConcreteStrategy that should be used. When an operation is required then the algorithm is run from the strategy object. The Context is not aware of the strategy implementation. If necessary, addition objects can be defined to pass data from context object to strategy.

The context object receives requests from the client and delegates them to the strategy object. Usually the ConcreteStrategy is created by the client and passed to the context. From this point the clients interact only with the context. In other words, it defines a family of algorithms, encapsulate each one and make them interchangeable. In computer programming, the strategy pattern also known as the policy pattern is a software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern

- defines a family of algorithms,
- encapsulates each algorithm, and
- makes the algorithms interchangeable within that family.

Strategy lets the algorithm vary independently from clients that use it. Strategy is one of the patterns included in the influential book Design Patterns by Gamma et al. that popularized the concept of using patterns in software design. For instance, a class that performs validation on incoming data may use a strategy pattern to select a validation algorithm based on the type of data, the source of the data, user choice, or other discriminating factors. These factors are not known for each case until run-time, and may require radically different validation to be performed. The validation strategies, encapsulated separately from the validating object, may be used by other validating objects in different areas of the system (or even different systems) without code duplication.The essential requirement in the programming language is the ability to store a reference to some code in a data structure and retrieve it. This can be achieved by mechanisms such as the native function pointer, the first-class function, classes or class instances in object-oriented programming languages, or accessing the language implementation's internal storage of code via reflection.

Structure



IV. Related Work

There are various design patterns that can be used to solve any of the industrial application. Here in this paper work, strategy pattern is used to build a framework. In accounting framework, one thing is mostly needed that is tax calculation. To solve this problem author in the current study has chosen the strategy pattern. Using these patterns, design solution of the industrial problem will be described. The father of the pattern concept, proposed a description template stating nine essential pattern elements. These patterns element describes the design patterns effectively; also describe how these patterns are useful to solve the problem. Industrial applications typically require different kinds of interfaces to the data they store and the logic they implement are data loaders, user interface and integration gateways and others. Instead of using for different purpose, these interfaces often need common interactions with the application to access and manipulate its data and invoke its business logic. These interactions may be complex, involving transaction across multiple resources and the coordination of several responses to an action. These interfaces decide the interaction between different layers of the application; how user interacts with middleware layer and the database layer. The framework is implemented in .Net. As we are using the design patterns to build this framework hence the developer can use this framework to build any kind of industrial application and can implement it in any other programming language using object-oriented concepts. Using the concept of design patterns. There are various classes with their methods and properties [5].

V. Analyze Strategy Pattern by Example

Strategy pattern is used when we have multiple algorithm for a specific task and client decides the actual implementation to be used at runtime.

Strategy pattern is also known as Policy Pattern. We defines multiple algorithms and let client application pass the algorithm to be used as a parameter. One of the best example of this pattern is Collections.sort() method that takes Comparator parameter. Based on the different implementations of Comparator interfaces, the Objects are getting sorted in different ways [8].

VI. Simulation Strategy Design Pattern

One common usage of the strategy pattern is to define custom sorting strategies e.g. to sort a list of strings by length in Java, passing an anonymous inner class (an implementation of the strategy interface) [7]:

```
List<String> names = Arrays.asList("Anne", "Joe",
"Harry");
Collections.sort(names, new Comparator<String>() {
public int compare(String o1, String o2) {
return o1.length() - o2.length();
}
});
Assert.assertEquals(Arrays.asList("Joe", "Anne",
"Harry"), names);
```

VII. Conclusion

Although the belief of utilizing design patterns to create better quality software is fairly widespread, there is relatively little research objectively indicating that their usage is indeed beneficial. In this paper we try to reveal the connection between design patterns and software maintainability. It is very hard to understand better what the patterns are and how they relate to each other. At this point there is a fundamental picture as reacting to an event to produce accounting entries. We used our probabilistic quality model for estimating the maintainability. We found that every introduced pattern instance caused an improvement in the different quality attributes. Moreover, the average design pattern line density showed a very high, 0.89 Pearson correlations with the estimated maintainability values. Design patterns are outstanding communication tool and help to make the design process faster. This allows solution providers to take the time to concentrate on the business implementation. Patterns help the design to make it reusable. Reusability not just applies to the component, but also the stages of the design that must go from a problem to final solution. The ability to apply a pattern that provides a repeatable solution is worth the little time spent learning formal patterns. There are some promising results showing that applying design patterns improve the different quality attributes according to our maintainability model. In addition, the ratio of the source code lines taking part in some design patterns in the system has a very high correlation with the maintainability. However, these results are only a small step towards the empirical analysis of design patterns and software quality [4].Design patterns shall support reuse of a software architecture in different application domains as well as reuse of flexible components [6].

References Références Referencias

1. http://blogs.infragistics.com/blogs/ux/archive/2009/02/03/what-is-a-design -pattern-and-why-use-them-for-quince.aspx.
2. Bertrand Meyer, Karine Arnout, Componentization: The Visitor Example, to appear in Computer (IEEE), 2006.
3. http://www.oodesign.com/abstract-factory-pattern.html
4. P´eter Heged˝us, D´enes B´an, Rudolf Ferenc, and Tibor Gyim´othy University of Szeged, Department of Software Engineering ´Arp´ad t´er 2. H-6720 Szeged,Hungary {hpeter,zealot,ferenc,gyimothy}-@inf.u-szeged.hu

5. Meyer, Bertrand "Componentization: The Visitor Example". *IEEE computer* (IEEE) 39 (7): 23–30.
6. Jurgen Dorn and Tabbasum Naz,Institute of Information Sysytems 184/2 Technicla University Vienna ,Favoritenstrabe 9-11, Vienna A-1040, Austria {dorn/naz}@dbai.tuwien.ac.at
7. http://stackoverflow.com/questions/370258/realworld-example-of-the-strategy-pattern
8. http://www.journaldev.com/1827/java-design-patterns-example-tutorial#strategy-pattern.