Designing Object Oriented Models through UML diagrams for Hotel Front Office System

Su Nandar Htay ; Khaing
University of Computer Studies, Yangon
sunandarhtay09@gamil.com,khaingaugust@gmail.com

Abstract

Design patterns are recurring solutions to software design problems you find again and again in real-world application development. Patterns are about design and interaction of objects, as well as providing a communication platform concerning elegant, reusable solutions to commonly encountered programming challenges. The proposed system deals with the object-oriented design model through Unified Modeling Language (UML) for a Front Office process of the Hotel. The FO process includes Room reservation, Check-in, accumulation of expense during the guest stay, Check out and Management reports. This system implements with the five design pattern classes. UML diagrams of Adapter, Singleton, Proxy, Visitor, Observer design patterns are used to build object-oriented models for developing the Hotel FO system. The designed objects are implemented to classes using C#.NET programming language.

1.Introduction

Design patterns document recurring solutions to recurring problems in object-oriented software design. They capture design expertise in reusable form. A design pattern has a name, a description of the problem it addresses, and a general solution that designers must tailor or their particular variant of the problem. The pattern also educates the reader about the consequences good and bad- of its application and about implementation variants. Design pattern do not describe new or novels designs; they are mined from successful object-oriented design. The patterns that we encounter need to be captured and documented in a sufficiently descriptive manner so that they can be referred for future use.

UML provides the perfect tools to do just this. The unified modeling Language (UML) is a very dominant modeling graphical language for specifying, constructing and documenting the artifacts of software system.

UML is a collection of best engineering practices that have successful in the modeling for a design of a huge and complex systems. Modeling is very important for readability and reuse of the systems.UML offers a set of notations and rules for using the same. The main task of the UML is to create simple, well documented and easy to understand software model for the people. In addition, UML has a sufficiently extensive and expressive vocabulary to capture the details of patterns. The UML modeling consists of nine main diagrams to model a software system. These diagrams are Use case Diagram, Class Diagram, Object Diagram, State Diagram, Activity Diagram, Sequences Diagram, Collaboration Diagram, Component Diagram and Deployment Diagram. The Class Diagram in UML can be used to capture the patterns in a system. The UML class Diagrams provide an easy way to capture and document Design patterns.

2.Related Work

Design patterns are commonly defined as time-tested solutions to recurring design problems. The term refers to both the description of a solution that you can read, and an instance of that solution as used to solve a particular problem. Design patterns have their roots in the work of Christoper Alexander, a civil engineer who wrote about his experience in solving design issues as they related to buildings and towns. It occurred to Alexander that certain design constructs, when used time and time again, lead to the desired effect. He documented and published the wisdom and experience he gained so that others could benefits. About 15 years ago, software professionals began to incorporate Alexander‘s principles into the creation of early design pattern documentation as a guide to novice developers. This early work led others to also write about design patterns and culminated in the publication of *Design Patterns:Element*

of Reusable Object-Oriented Software in 1995 by Eric Gamma, Richard Helm, Ralph Johnson, and John Vlissides. This book is considered to be the "coming out" of design patterns since. Design Patterns described 23 patterns that were based on the experience of the authors at that time. These patterns were selected because they represented solutions to common problems in software development. Many more patterns have been documented and cataloged since the publishing of *Design Patterns*. However, these 23 are probably the best known and certainly the most popular.Design patterns are represented as relationships between classes and objects with defined responsibilities that act in concert to carry out the solution. To illustrate a design pattern, consider the Bridge pattern, one of the original 23 patterns described in *Design Patterns.* One component that is not shown in the class diagram is a database accessing that will provide database accessing services for all the classes in the class diagram that need to interact with a data store. All database interactions to retrieve or update information required for their business functionality can be routed through this database access element, which acts as a bridge between the database and the application components. Adapter provides a solution to the scenario in which a client and server need to interact with one another, but cannot because their interfaces are incompatible. To implement an Adapter, you create a custom class that honors the interface provided by the server defines the server operations in terms the clients expects. This is a much better solution than altering the client to match the interface of the server.

3.Theory Background

3.1The Unified Modeling Language

A picture is worth a thousand words, this absolutely fits discussing about UML.Object oriented concepts were introduced much earlier than UML. So at that time there were no standard methodologies to organize and consolidate the object oriented development.

At that point of time UML came into picture. The unified modeling is the successor to the wave of object-oriented 0061nalysis and design methods that appeared in the late 80's and early 90'.It most directly unified the methods of Booch,Rumbaugh(OMT) and Ivar Jacobson. It is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. It is a very important part of developing object - oriented software and the software development process. It is intended for use with all development methods, life cycle stages, application domain and media. UML captures information about the static structure and dynamic behavior of a system. It uses mostly graphical notations to express the design of software projects. Its diagrams are not only made for developers but also for business users, common people and anybody interested to understand the system. Using the UML diagrams helps project teams communicate, explore potential designs, and validate the architectural design of the software.

3.1.1 UML Diagrams

Diagrams are the heart of **UML**.These diagrams are broadly categorized as structural and behavioral diagrams.

Structural diagrams are consists of static diagrams like class diagram, object diagram etc.

Behavioral diagrams are consists of dynamic diagram like sequence diagram, collaboration diagram etc.

The static and dynamic nature of a system is visualized by using these diagrams.

- **A class diagram** shows a set of classes, interfaces, and collaborations and their relationships. These diagrams are the most common diagram found in modeling object-oriented systems. Class diagrams address the static design view of a system.
- **An Object diagram** is an instance of a class diagram. So the basic elements are similar to class diagram. It consists of object and links.
- **A component diagram** shows the organizations and dependencies among a set of components. It addresses the static implementation view of a system.

  A deployment diagram shows the configuration of run-time processing nodes and the components that live on them. It addresses the static deployment view of architecture.
- **A use case diagram** shows a set of use cases and actors and their relationships. It is used to capture the dynamic nature of

the system. So it represents the system functionalities and their flow.

- **An interaction diagram** is used for capturing dynamic nature of a system. Sequence and collaboration diagrams are kinds of interaction diagrams. A sequence diagram emphasizes the time-ordering of messages. A collaboration diagram emphasizes the structural organizations of the objects that send and receive messages.
- **A state chart diagram** shows a state machine, consisting of states, transitions, events, and activities. It addresses the dynamic view of a system.
- **An activity diagram** is a special kind of statechart diagram that show the flow from activity to activity within a system. It addresses the dynamic view of a system.

3.2 Design Pattern

Design Pattern makes it easier to reuse successful designs and architectures. Expressing proven techniques as design patterns makes them more accessible to developers of new systems. Design patterns help you choose design alternatives that make a system reusable and avoid alternatives that compromise reusability. Design patterns can even improve the documentation and maintenance of existing systems by furnishing an explicit specification of class and object interactions and their underlying intent. Put simply, design patterns help a designer get a design "right" faster.

In general, a pattern has four essential elements:

- The pattern name is a handle we can use to describe a design problem, its solutions, and consequences in a word or two. Naming a pattern immediately increases our design vocabulary. Having a vocabulary for patterns lets us talk about them with our colleagues, in our documentation, even to ourselves. It makes it easier to think about designs and to communicate them and their trade-offs to others. Finding good names has been one of the hardest parts of developing our catalog.
- The problems describe when to apply the pattern. It explains the problem and its context. It might describe specific design problems such as how to represent algorithms as objects. It might describe class or object structures that are symptomatic of an inflexible design. Sometimes the problem will include a list of conditions that must be met before it makes sense to apply the pattern.
- The solution describes the elements that make up the design, their relationship, responsibilities, and collaborations. The solution doesn't describe a pattern is like a
- template that can be applied in many different situations. Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements solves it.
- The consequences are the results and trade-offs of applying the pattern. Though consequences are often unvoiced when we describe design decision, they are critical for evaluating design alternatives and for understanding the costs and benefits of applying the pattern. The consequences for software concern space and time trade-offs. Since reuse is often a factor in object-oriented design, the consequences of a pattern include its impact on a system's flexibility, extensibility, or portability. Listing these consequences explicitly helps you understand and evaluate them.

3.2.1 GOF Design Pattern

The Gang of Four(GOF) patterns are generally considered the foundation for all other patterns. They offer flexible solutions to common software development problems. Each pattern is comprised of a number of parts, including purpose/intent, solution structure and implementations. They are categorized in three groups: Creational, Structural, and Behavioral.

Creational Patterns	
Abstract Factory	Creates an instance of several different classes
Builder	Separates object construction from its representation
Factory Method	Creates an instance of several derived classes
Prototype	A fully initialized instance to be copied or cloned
Singleton	A class to allow only a single instance

Behavioral Patterns	
Chain of Responsibility	A way of passing a request between a chain of objects
Command	Encapsulate a command request as an object
Interpreter	A way to include language elements in a program
Iterator	Sequentially access the elements of a collection
Mediator	Defines simplified communication between classes
Memento	Capture and restore an object's internal state
Observer	A way of notifying change to a number of classes
State	Alter an object's behavior when its state changes
Strategy	Encapsulates an algorithm inside a class
Template Method	Defer the exact steps of an algorithm to a subclass
Visitor	Defines a new operation to a class without change

Structural Patterns	
Adapter	Match interfaces of different classes
Bridge	Separates an object's interface from its implementation
Composite	A tree structure of simple and composite objects
Decorator	Add responsibilities to objects dynamically
Facade	A single class that represents an entire subsystem
Flyweight	A fine-grained instance used for efficient sharing
Proxy	An object representing another object

4. Implementation of Hotel Front Office System

The proposed system implements with five design pattern classes. It has used Creational Pattern that includes Singleton Pattern, Structural Pattern that includes Adapter and Proxy Patterns, and Behavioral pattern that includes Visitor and Observer Patterns.

4.1 System Flowchart for Hotel Front Office System

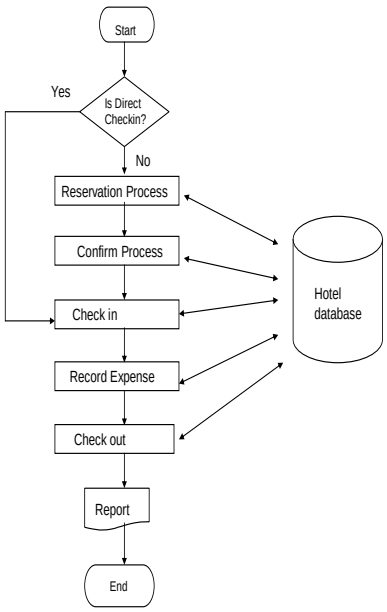

Figure1: System Flowchart for Hotel Front Office System

The flowchart of the Hotel Front Office System can be seen as shown in figure, there are five main process and it includes report stage.

4.1.1Reservation Process

When the guest reserves the room, the guest's information and the room that the guest want to stay are recorded by the FO staff using the Reservation form. Then, the related information of the guest is added to the reservation list form.

The room is assigned from the room status list form that shows the status, price, facilities and type of the room. The room status such fill or free is changed due to the visitor design pattern. The necessary information of the room list can be processed by the Adapter design pattern.

4.1.2Confirmation Process

The guest informs the FO staff that she will stay or cancel or change before 1 weeks of

arrival. If the guest confirm to stay, the guest from the reservation list is added to the confirm list form. If he wants to cancel the reservation, his information from the reservation list form can be deleted .

4.1.3Check-in process

The additional information of the guest is recorded. Then it is added to the check-in list.

4.1.4 Expense Process

The expense of the rooms such as phone, extra bed, car rental, internet & email etc... Are recorded daily by using the expense list form. This form shows the expense type, rate, total amount, unit of each expense at specific date.

The addition of each expense type and rate to the expense type form is done by the Singleton design pattern.

4.1.5 Check-Out process

The total charges of the room and extra charges are calculated. If the guest who is member of the hotel, he will get discount .If he gets expired date, he will not get this discount. The guest who is not member can get discount as required. The members of the hotel can be registered from the member form. This member will be updated according to the member life and fees on the member type form. This process is done by the observer design pattern.

When the currency change is required, the total amount of the room is calculated on the exchange rate. This process is done by the Proxy design pattern. After doing the check-out process, the room status will become free.

4.1.6 Report

- **Guest List Report** shows the guest information during the guest stay.
- **Room List Report** shows the status of room such as fill or free.
- **History Report** shows the information of the guest who have been checked out during the limited period.

4.2 Overall Design Pattern Classes' Implementation

4.2.1Singleton Design Pattern

Ensure a class has only one instance and provide global of access to it.Several different client objects need to refer to the

Expense and to ensure that clients do not have more than one of them.

\

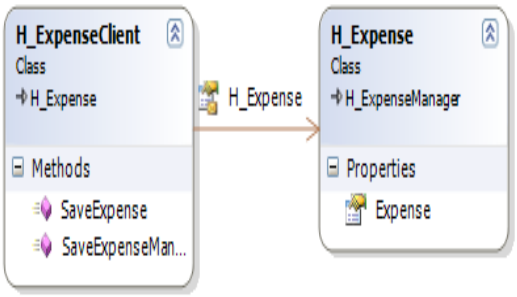

Figure2: UML Class diagram for Singleton

4.2.2 Proxy Design Pattern

Provide a surrogate or placeholder for another object to control access to it. A proxy pattern constitute for the actual object. Use of proxy is prevalent in remote object interaction protocols.In check out process,
H- CurrencyChange class acts as a substitute for the H-CheckOut class.

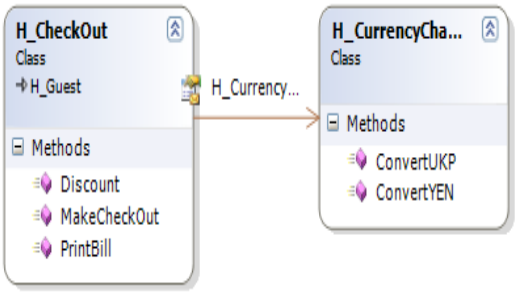

Figure 2:UML Class diagram for Proxy

4.2.3Visitor Design Pattern

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates. Visitor: The status of the room is changed by the hotel Processreservation class.

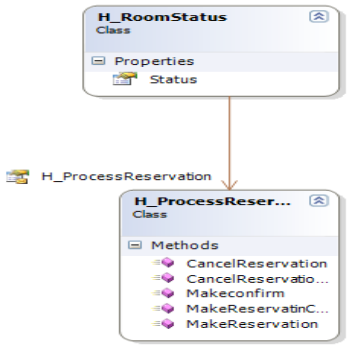

Figure 4: UML Class diagram for Visitor

4.2.4 Adapter Design Pattern

Convert the interface of a class into another interface clients expect. Adapter lets

classes work together that couldn't otherwise because of incompatible interface. Adapter: the H-Roomoperation class acts as adapter between H_Room class and H_RoomType class ,H_RoomStatus class.

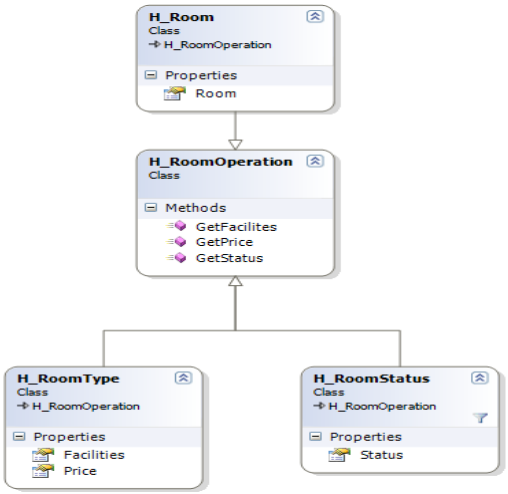

Figure 5: UML Class diagram for Adapter

4.2.5 Observer Design Pattern

Define a one to many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.Observer: When hotel Member type class changes something, its dependent classes are updated automatically.

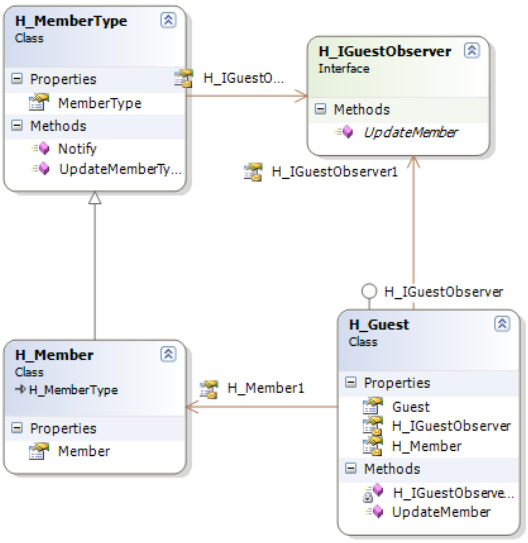

Figure 6:UML Class diagram for Observer

5. Conclusion

This paper show the efficiency of design pattern using in UML models to develop the object based business application system. Design patterns integration into a modeling language is a tempting idea. A simple modeling constructs allowing to explicitly point out participant classes of a design pattern could help designers in many ways. Besides the direct advantage of a better documentation and the consequent better understandability of a model, pointing out the existence of a design patterns allows designers to abstract known design details (e.g. associations, methods) and concentrate on more important tasks.

By using design pattern in my system, it can be reduced the designing time and more importantly ensure that the system is constistent and stable in terms of architectural and design.

6.References

[1] Unified Modeling Lanuage
Mark Priestley,"Practical Object-Oriented Design With UML

[2] Design Pattern
Su Hnin Aye
Implementing Design Pattern For Transformer Sales

[3] Unified Modeling Language
http://www.atlas.kennesaw.edu

[4]http://www.developer.com/design/article.php

[5] Gang of four design pattern
www.dofactory .com