Object-oriented Composition Mechanismsin Unified ModellingLanguage

Ya Min Thu
University of Computer Studies, Banmaw
yaminthu.10.9@gmail.com

Abstract

This paper presents a new concept of inheritance and association that supports using object-oriented composition. In object-oriented programming, composition can be used, if an object has one object or is part of another object, for instance, if a computer has a motherboard; the motherboard is a computer component. This system allows the users to define the links that connect objects and express and combine individual composition properties. This system can create class diagram in which field, method, constructor and destructor can be added. Class diagrams can be associated and inherited with each other and generate codes. Abstract class can be created and also associated and inherited with other classes. The system generates code and displays composition techniques and properties.

Keywords: *Class, Object, Object-oriented programming, inheritance, association*

1. Introduction

Composition is the basic concepts in object-oriented programming. It shows a class that references one or more objects of other classes, for example, variables. This allows user to model a Has-A association between objects.Association is relation between two separate classes which establishes through their Objects. There are several association models. They are one-to-one, one-to-many, many-to-one, and many-to-many. In OOP, an object communicates with anotherobject to use functionality and services provided by that object vice visa. The two types of association are Composition and Aggregation.

Applications are able to build more quickly due to the reusable of object. There is a gap between software written in an object-oriented language and a reusable framework of object classes. Frameworks are hard to develop, and not always easy to use. UML assumes a process that is use-case driven, architecture-centered, iterative and incremental (Bahrami, 1999). It can solve many software architecture issues and capture the static and dynamic structure and collaboration among key participants in software designs. Moreover, it focuses on tight scope.

UML is connected with object oriented design and analysis. It provides the use of elements and form associations between them to form diagrams. Diagrams of UML can be classified as Structural Diagrams and Behavior Diagrams.

Object-oriented software composition adopts the viewpoint that object-oriented technology is essentially about composing flexible software applications from software components. Although object-oriented languages, tools and methods have come a long way since the birth of object-oriented programming, the technology is not yet mature [1].

In this paper, object-oriented composition and Uniformed Modelling Language areintroduced. Section 2 presents the related works. Section 3 discussesproposed system. Section 4 includes specifying object implementations. Inheritance and association are presented in section 5. The conclusion of this system combines at the last section 6.

2. Related Works

A new reference to psychoactive compounds for object reference provides a clear linguistic approach to show and combine the properties of each of the elements we need. It allows programmers to smoothly represent the radiance of the meaning of an element over a period of time between the element and the heredity of the object. Understand the resulting program better because it is less likely to occur due to design decisions and detailed changes. They are controversial; Referrals Acquisition [8].

. Converting class libraries is a difficult or impossible way to modify class libraries by exchanging class libraries, so changing class libraries can change class libraries. Inherit a class in the application program. Introduce legacy type interfaces, which allow secure type changes for legacy modelling collections with clear links and parameters. In addition, this approach opens up new possibilities for configuration and simplifies programming [9].

3. Proposed System

In object-oriented composition tool, the user can draw class diagrams in which field, method, constructor and destructor can be added. And then, the user can connect class diagrams with each other. The system generates code automatically and display composition techniques and properties.

There are many composition techniques. But the system can use two composition techniques; inheritance and association. Inheritance is a method for composition between objects. The association is a way of describing that a class knows about and holds a reference to another class. There are different composition properties, such as overriding, redirection, acquisition

and polymorphism. Overriding a method means replacing the superclass's implementation of a method with one of subclass's implementation.Redirection refers to the current value of the reference inside the object. Object-oriented composition mechanisms can display step-by-step instructions.

The difference between object-oriented composition mechanisms and the other CASE tools is that the system takes class diagram as input and generates source code and displays composition techniques and properties. Many CASE tools are available now. For instance, reverse engineering can be used to feed source code into a specification tool for structural analysis, graphics, and design models. List applications and other design data, tools, reorganization and code analysis, program syntax analysis, control flow curves and create systematic automation [4].

3.1. Using Object-Oriented Method

The approach to using object-oriented methods in system design is called object-oriented design. The object-oriented development approach is to create them into common computer applications; systems that use emerging material technologies to manage and collect data. Object-Oriented Design is the study of object-oriented analysis [10].

4. Specifying Object Implementations

The implementation of an object is determined by its class. The class defines the internal data and representations of the objects. The object defines the actions that can be performed. The class is shaped like a rectangle with the class name. Actions will appear in a regular category under the data class name. Any class comes after the action. Class name dividing line from data from operations and operations [11].

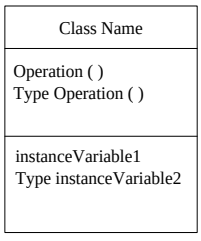

Figure 1. Example of a class diagram

The return type and the variable instance type are optional. The object is created by creating an object class instance as an example of the class. The instance class creation process allocates the internal storage of the object. Objects (consisting of instance variables) and associating operations with these data can create similar instances of the object by creating an instance class.

The new class can be defined in terms of existing classes using class inheritance. When a subclass inherits from the parent class, the class includes all the definitions and operations that the parent class defines as an object. The instance of a subclass contains all the information defined by the subclass and the parent class, and can perform all the functions defined by this subclass and the parent. Subclass relationship is represented as vertical line and triangles [5].

Subclasses can customize and define the behaviour of the parent class; in particular, the class may override the execution defined by the parent class. Inherit a simple class assignment class by extending other classes, making it easy to define a family of objects with related functions [12].

5. Composition Techniques of the System

5.1. UML Inherited Property

Classes are abstraction in object-oriented modelling and object-oriented programming languages. As with psychiatric data types, it covers both class structure and behaviour. Different types of mental data can be defined by other classes by inheriting classes. [3] UML has an equal general relationship with arrow headed moving from subclasses within the line to the super head. Another salesperson shows the differences between the arrowhead types used in UML-style relationships and the users in UML-linked directories. Another difference is that, unlike organizations, there is generally no need for public names at the end of a relationship. UML describes the inherited team as part of a designated parent class. It works like a parent but he has always been discriminated against. Defines an attribute of an inherited member. Inherited categorization events (or properties in general) can include values or collections.
We have a watch that can be used as a calculator. And then there is the CalculatorWatch class. It is seen as a revision of the Watch class. This type of relationship between the base class and the modified class is called inheritance. A class is generally called a superclass. The class (CalculatorWatch) is called a subclass. In an inherited relationship, The subclass is shown in Figure 2; Purify the superclass by defining the attributes and functions in CalculatorWatch. Defines functionality for simple arithmetic operations without conventional clocks. Superclass and subclass are related terms. The same class can be a subclass associated with one class and a superclass associated with another [13].

Inherited relationships include shared models; the result is called an abstract class, unless it is intended to be instantiated in general only if it is performed only on attributes and operations. Psychological classes often represent general concepts in the application domain and display their names in italics. Inheritance relationships are represented by triangles and lines. The triangle points to the superclass and connects the other end subclass..

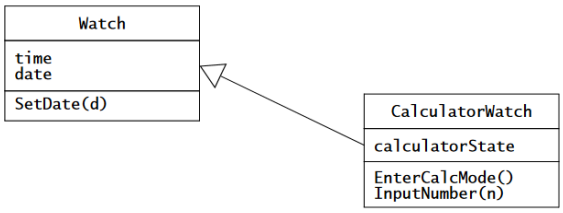

Figure 2. Subclass relationship

5.2. Association

Similar objects are grouped together and represented by a single row. The connection is represented by a structure known as a relay [6]. Linking between two objects and the model connects some types of connections between objects. Usually the concept expressed by association can be described as a common relationship between the classes involved. So the team consists of several classes. Links are organized into a line and teams are displayed in UML [14].

System Design

The use case diagram is used for object oriented composition tool to draw class and generate code and its properties. There are six use cases. The user can do the process of

- Draw class diagram
- Add methods
- Connect class
- Generate code
- Display composition techniques
- Display composition properties.

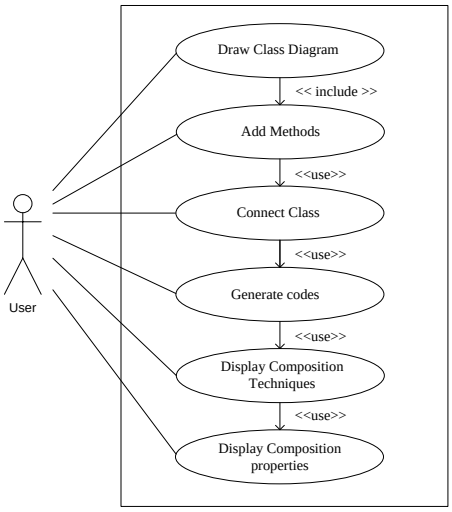

Figure 3. Use case diagram

Composition Scenario: The Account Example

The system can be used for many projects such as account, booking system, library system, shopping, university, etc. to generate code. Consider an application in the field of banking with accounts and permanent orders. The relationship between an individual / company and an accountant is often intertwined. But in this example, each account requires a specific role for the owner.

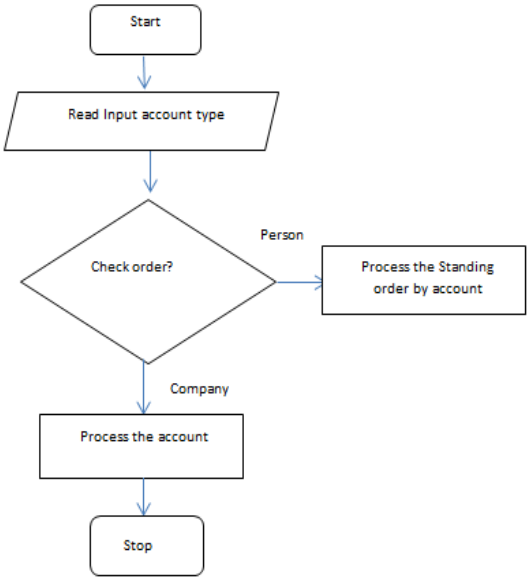

Figure 4. System flow diagram for account example

Forinstance, the above figure 4, the system flow diagram for account example is shown in one person has only one legacy account and the company has multiple accounts (savings, current accounts) and accounts may change frequently.Shared Class One Standing Order Processing (SOP) is used to deregister and execute standing orders[2].

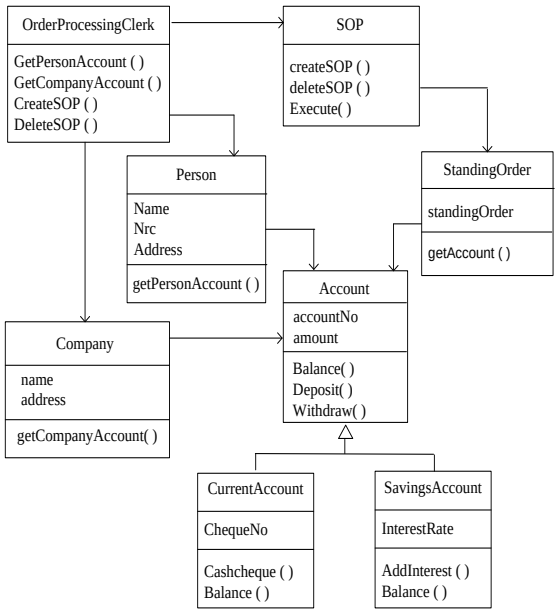

Figure 5. Class diagram for account example

The class diagram for this problem is shown in Figure 5 and this figure shows from the information about money orders, the order processing clerk receives the accounting object from the relevant person / company, creates a standing order, and registers it with a SOP.This design is simple and easy to understand.

Output Code for Account

```
public class Account
 {
        public static int accountNo;
        public static int amount;
        public static void balance ( )
        {
        }
        public static void deposit ( )
        {
        }
        public static void withdraw ( )
        {
        }
} etc.
```

Overriding for Account

One way to override is to override the implementation of the superclass by using a subclass. Signatures must be the same. The override method has its own access identifier. Subclasses can change the method of the superclass. But we will be more clearly visible. You can delete only one method. If this method is not available, it is not inherited. If it is not inherited, delete it. Not available Erasing is a structural property.

In Figure 5, the user requests a method. GetPersonAccount () in the OrderProcessingClerk class. To retrieve the balance of the savings account in the class, add the GetPersonAccount () and balance () methods in SanvingsAccount () and add them from the account class.

Redirection for Account

Redirection is a property of an element. The user requested a method. In the first OrderProcessingClerk class, call getPersonAccount () and then the method. Enter getPersonAccount () from the person class and call the method in the Account class. The user requested the createSOP () method in the OrderProcessingClerk class. Then call the register () method in the SOP class and the getAccount () method in the Standing Order class to create a status sequence for the selected account. There is a change in the Account class. Therefore, the getPersonAccount () method often refers to the current value of the current reference without individual objects.

Example of Inheritance relationship

Consider the example shown in Figure 6. There are three types of employees: Consult with HourlyEmployees, SalariedEmployees. Features shared by all employees: empName, empNumber, address, dateHired, and printLabel -are stored in the Employee superclass. The attributes of each type of employee are stored in the respective subclasses (eg hourlyRate and HourlyEmployee computeWages)

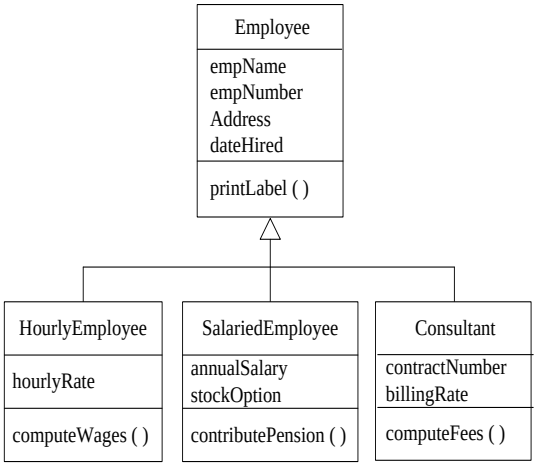

Figure 6.Employee superclass with three subclasses

An inheritance is represented as a solid line from subclass to superclass. The subclass inherits all the features of its superclass, such as the hourlyRate and computeWages custom functions, as well as the HourlyEmployee type, inherited by employame, empNumber, address, dateHired, and printLabel. An example of an HourlyEmployee would store values for Employee and HourlyEmployee properties. When prompted, printLabel and computeWages operations will be used[7].

This system generates codes for Figure 6.

```
public class Employee
 { public static string empName;
        public static string empNumber;
        public static string address;
        public static string dateHired;
        public static void printLabel ( )
        {
        }
}
public class HourlyEmployee: Employee
 {
        Employee myEmployee=new Employee();
        public static int hourlyRate;
        public static void computeWages ( )
        {
        }
}
public class SalariedEmployee: Employee
 {
        Employee myEmployee=new Employee();
        public static int annualSalary;
        public static string stockOption;

        public static void contributePension ( )
        {
        }
}
```

```
public class Consultant: Employee
 {
        Employee myEmployee=new Employee();
        public static int contractNumber;
        public static int billingRate;
        public static void computeFees ( )
        {
        }
}
```

Example of Association relationship

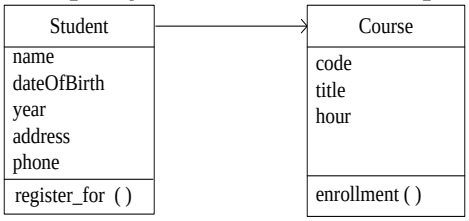

Figure 7.Example of association relationship

This system generates codes for Figure 7.

```
public class Student
 { public Course course
        {
                get
               {
               }
               set
               {
               }
        }
        public static string name;
        public static string dateOfBirth;
        public static string year;
        public static string address;
        public static int phone;
        public static void register_for ( )
        {
        }
}

public class Course
 {
        public static string code;
        public static string title;
        public static string hour;
        public static void enrollment (  )
        {
        }
}
```

6. Conclusion

This system can be able to create a new class that may be an extension or specialization of an extension class. In addition, this system can modify the implementation of the system without difficulty using class inheritance. This system can create class diagram in which field, method, property, constructor, destructor can be added. Hence, class diagrams can also be connected with each other by association and inheritance, and then generate codes.The composition techniques can be added to this system to be complete. This system can enhance codes that are added to complete the data and message passing between classes to run the application in the future.

Acknowledgment

My sincere thanks to Dr. May Phyo Oo to her encouragement in writing research paper. Thanks to my friends for putting up with me, and giving lots of very useful advice.

References

[1] O. Nierstrasz and D. Tsichritzis, "*Object-oriented Software Composition"*, ISBN 0-13-220674-9, Prentice Hall Object Oriented Series, 1995

[2] K. Ostermann and M. Mezini, "Object-oriented Composition Untangled",*Siemens AG, Corporate Technology SE 2, D81730*, Munich, Germany

[3] http://www.omg.org,Object Management Group, *OMG Unified Modeling Language Superstructure.Version 2.2*.

[4] R. S. Pressman, "*Software Engineering"*, Fifth Edition , McGraw Hill

[5] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software"*, Produced by KevinZhang

[6] M. Priestley, "*Practical Object-Oriented Design with UML"*, ISBN 007-123923-5, Second Edition, McGraw Hill, International Edition 2004.

[7] J. S. Valacich, J. F. George and J. A. Hoffer, " *Essentials of Systems Analysis and Design"*, ISBN 0-13-201756-3, Third Edition, Prentice Hall

[8] G. Booch, "*Object-Oriented Analysis and Design with Applications"*, 2nd ed.,Benjamin/Cummings, Redwood City, CA, 1994.

[9] G. Booch, J. Rumbaugh, & I. Jacobson, "*The Unified Modeling Language UserGuide"*, Addison-Wesley, Reading, MA, 2005.

[10] P. Coad, D. North, & M. Mayfield, "*Object Models: Strategies, Patterns, &Applications"*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[11] L.L. Constantine & L.A.D. Lockwood, "Structure and style in use cases for userinterface design", in M. van Harmelen (ed.), 2001.

[12] T. De Marco, "*Structured Analysis and System Specification"*, Yourdon, New York,1978.

[13] B.P. Douglass, "*Doing Hard Time Using Object Oriented Programming andSoftware Patterns in Real Time Applications"*, Addison-Wesley, Reading, MA, 1999.

[14] M. Fowler, *UML Distilled: "A Brief Guide To The Standard Object ModelingLanguage"*, 3rd edition, Addison-Wesley, Reading, MA, 2003.