OBJECT-ORIENTED COMPOSITION TOOL

Phyu Thwe, San San Tint
University of Computer Studies, Mandalay
phyuthwe19@gmail.com

ABSTRACT

Object-oriented language comes with pre-defined composition mechanisms, such as inheritance, object composition or delegation each characterized by a certain set of composition properties. The concepts of inheritance and association are used in object-oriented composition. 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.

1. INTRODUCTION

Object-oriented programming is a better kind of structured programming perhaps, but structured programming methods would not help very much in developing object-oriented applications. Object-oriented programming is not just structured programming. Applications are able to build more quickly because objects are reusable - there can be a huge gap between software written in an object-oriented language and a truly reusable framework of object classes. Frameworks are hard to develop, and not always easy to use.

There are good reasons for adopting object-oriented technology: so far it appears to offer the best means to cope with complexity and variation in large systems. When families of similar systems must be built, or single system must undergo frequent changes in requirements, object-oriented languages, tools and methods offer the means to view such systems as flexible composition of software components. It may be still requiring a great deal of skill to build flexible systems that can meet many different needs, but at least object-oriented technology simplifies the task. 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 tool is implemented by using classes and objects. The rest of the paper describes as follows: section 2 presents the related works. Section 3 includes proposed system. Section 4 includes specifying object implementations. Inheritance and association are presented in section 5. Section 6 fulfills with the design and implementation of the system. The conclusion of this system combines at the last section 7.

2. RELATED WORKS

Compound references, a new abstraction for object references that allows us to provide explicit linguistic means for expressing and combining individual composition properties on-demand [2]. The model is statically typed and allows the programmer to express a seamless spectrum of composition semantics in the interval between object composition and inheritance. The resulting programs are better understandable, due to explicitly expressed design decisions, and less sensitive to requirement changes. They discuss the set of composition properties of overriding, redirection, acquisition, subtyping and polymorphic.

An approach to typed inheritance relationships allows a type-safe exchange of

classes in class libraries [3]. In which maintenance of class libraries becomes difficult or even impossible because changes in a class library can cause changes in the inheriting classed in an application program. Modeling inheritance with explicit and parametrical bindings introduces a type interface for inheritance which allows type-safe changes in class libraries. Furthermore the approach opens new possibilities for composition and makes programming easier.

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 always refers to the current value of the reference within an object. This system can display overriding and redirection.

The difference between this system 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 example, Reverse engineering to specification tools takes source code as input and generates graphical structured analysis and design models, where-used lists, and other design information. Code restructuring and analysis tools analyze program syntax, generate a control flow graph, and automatically generate a structured program [4].

The advantages of the system are to give the ability to create a new class that is an extension or specialization of an extension class and to modify the implementation of the system easier using class inheritance. The user can save time for writing source code by using this system.

4. SPECIFYING OBJECT IMPLEMENTATIONS

An object's implementation is defined by its class. The class specifies the object's internal data and representation and defines the operations that object can perform. A class is depicted as a rectangle with the class name. Operations appear in normal type below the class name. Any data that the class defines comes after the operations. Lines separate the class name from the operations and the operations from the data [5].

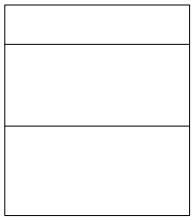

Figure 1. Example of a class diagram

Return types and instance variable types are optional. Objects are created by instantiating a class. The object is an instance of the class. The process of instantiating a class allocates storage for the object's internal data (made up of instance variables) and associates the operations with these data. Many similar instances of an object can be created by instantiating a class.

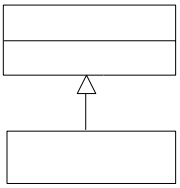

Figure 2. Subclass relationship

New classes can be defined in terms of existing classes using class inheritance. When a subclass inherits from a parent class, it includes the definitions of all the data and operations that the parent class defines. Objects that are instances of the subclass will contain all data defined by the subclass and its parent classes, and they will be able to perform all operations defined by this

subclass and its parents. The subclass relationship is indicated with a vertical line and a triangle [5].

Subclasses can refine and redefine behaviors of their parent classes. More specifically, a class may override an operation defined by its parent class. Overriding gives subclasses a chance to handle requests instead of their parent classes. Class inheritance defines classes simply by extending other classes, making it easy to define families of objects having related functionality.

5. COMPOSITION TECHNIQUES OF THE SYSTEM

5.1 Inheritance

A characteristic feature of object-oriented programming is inheritance. Inheritance is a method for composition between classes rather than between objects [3]. The extendibility as well as the reusability of software components is enhanced if the concept of inheritance is incorporated. Inheritance is the means by which objects of a class can access member variables and functions contained in a previously defined class, without having to restate those definitions. This system can give the ability to create a new class that is an extension or specialization of an extension class.

5.2 Association

Similar objects are grouped together and described by a single class; related links are described by a single construct, known as an association [6]. A link between two objects models some sort of connection between the linked objects. Normally, the idea expressed by a link can be described as a more general relationship between the classes involved. An association therefore involves a number of classes and models a relationship between the classes. Associations are represented in UML as lines joining the related classes.

6. 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.

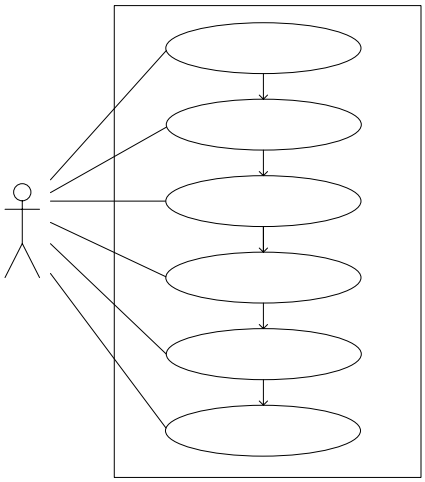

Figure 3. Use case diagram

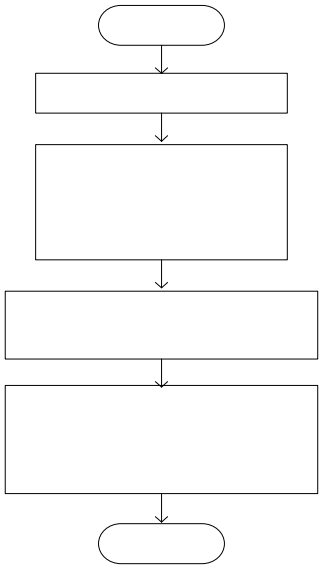

Figure 4. System flow diagram

6.1. 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 banking domain with persons, companies, accounts, and standing orders. The relation between persons/companies and accounts is usually one to many. However, in this example each account wants to have a dedicated role for its owner.

In this example, a person has only one main account and a company has an account. Different kinds of accounts exist (Savings Account, Current Account), and accounts are subject to frequent changes at runtime. A particular account may be shared. A class Standing Order Processing (SOP) is used for the registration deregistration and execution of standing orders [2].

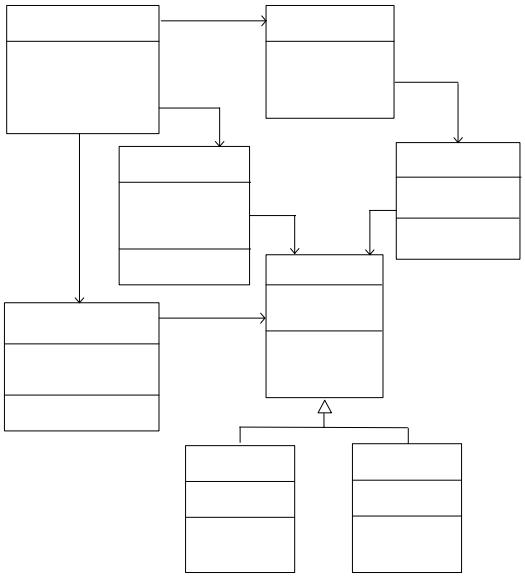

Figure 5. Class diagram for account example

A class diagram for this problem is shown in Figure 5. Based on the information on a pay order, the Order Processing Clerk gets the account objects from the involved Person/Company, creates a Standing Order and registers it with a SOP. This design is simple and easy to understand.

6.2. 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.
```

6.3. Overriding for Account

Overriding a method means replacing the superclass's implementation of a method with one of subclass's implementation. The signature must be identical. The overriding methods have their own access specifiers. A subclass can change the access of a super class's methods, but only to provide more access. A method can be overridden only if it is accessible. If the method is not accessible then it is not inherited, and if it is not inherited, it can't be overridden. Overriding is one of the composition properties.

In Figure 5, to get the saving account's balance, the user requests the GetPersonAccount ( ) method in the OrderProcessingClerk class and invokes the GetPersonAccount ( ) in person class and the balance ( ) methods in SanvingsAccount by overriding the method in Account class.

6.4. Redirection for Account

Redirection is one of the composition properties. The user first requests the getPersonAccount ( ) method in the OrderProcessingClerk class. And then invokes the getPersonAccount ( ) method in Person class and executes the methods in Account class. The user requests the createSOP ( ) method in the OrderProcessingClerk class and then invokes the register ( ) method in SOP class and the getAccount ( ) method in Standing Order class to

OrderProcessingClerk

GetPersonAccount ( )
GetCompanyAccount ( )
CreateSOP ( )

SOP

createSOP ( )
deleteSOP ( )
Execute( )

create the standing order for the selected Account. There are changes in Account class. So the getPersonAccount ( ) method always refer to the current value of the account reference without a Person object.

6.5. Example of Inheritance relationship

Consider the example shown in Figure 6. There are three types of employees: HourlyEmployees, SalariedEmployees, and Consultants. The features that are shared by all employees - empName, empNumber, address, dateHired, and printLabel - are stored in the Employee superclass, whereas the features to a particular employee type are stored in the corresponding subclass (e.g, hourlyRate and computeWages of HourlyEmployee).

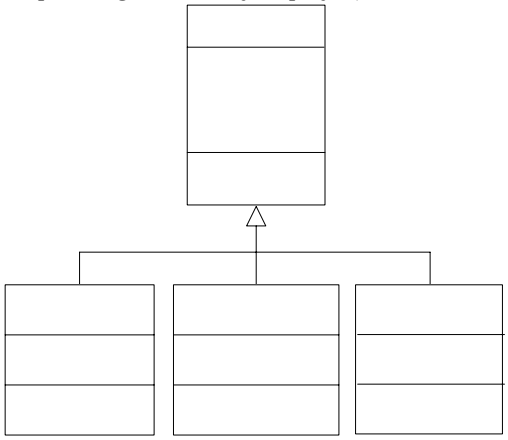

Figure 6. Employee superclass with three subclasses

An inheritance is shown as a solid line from the subclass to the superclass, with a hollow arrowhead at the end of, and pointing toward, the superclass. A subclass inherits all the features from its superclass. For example, in addition to its own special features - hourlyRate and computeWages - the HourlyEmployee subclass inherits empName, empNumber, address, dateHired, and printLabel from Employee. An instance of HourlyEmployee will store values for the attributes of Employee and HourlyEmployee and, when requested, will apply the printLabel and computeWages operations [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 ( )
        {
        }
}
```

6.6. Example of Association relationship

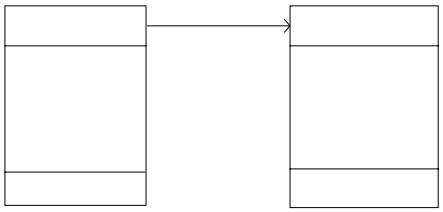

Figure 7. Example of association relationship

This system generates codes for Figure 7.

Employee

empName
empNumber
Address
dateHired

printLabel ( )

```
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 (  )
          {
          }
}
```

7. CONCLUSION

In this paper, the traditional object-oriented composition mechanisms are shown. A characteristic feature of object-oriented programming is inheritance. Inheritance is often regarded as the feature that distinguishes object-oriented programming from other modern programming paradigms and many of the alleged benefits of object-oriented programming, such as improved conceptual modeling and reusability. By means of inheritance, object-oriented programming enables the extension of components without losing compatibility.

The composition mechanism is similar to functional composition. This system can create class diagram in which field, method, property, constructor, destructor can be added. Class diagrams can be connected with each other by association and inheritance, and then generate codes.

This system is only to generate codes but is not able to run. This system cannot examine the syntax error, so the user must input data correctly. There are many relationships, but this system uses association and inheritance. This system cannot support message passing.

There are some areas of future work. The composition techniques can be added to this system to be complete. This system can be enhanced that codes are added to complete the data and message passing between classes to run the application.

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] F. J. Hauck, "Inheritance Modeled with Explicit Bindings: An Approach to Typed Inheritance," *University of Erlangen-Nurnberg, IMMD 4*, Germany

[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