Mapping of UML Diagrams to Executable Code

Shugang Liu [1, a], Jinfeng Chen [2,b], Yifei Liu[3,c] and Pengrui Lv[4,d]

[1, 2] Department of Computer Science. North China Electric Power University. Baoding. Hebei Province, China

[3] Institute of Political Science and Law. Taiyuan University of Technology. Taiyuan. Shanxi Province, China

[4] Science and Technology College. North China Electric Power University. Baoding. Hebei Province, China

[a] lsg69@qq.com, [b] 1281728917@qq.com, [c] 342450017@qq.com, [d] 827145707@qq.com

Keywords: object-oriented modeling; activity diagram; state diagram; code.

Abstract: As the standard object-oriented modeling language, UML describes object-oriented systems from all angles and is object-centric. It is unified with the most popular object-oriented programming languages and can generate code frameworks through class diagrams. There is a corresponding relationship from UML diagram to the object-oriented executable code. By finding the mapping algorithm, it can make the graphics generate more complete code and improve the efficiency of software development and code quality.

Introduction

As computer technology advances, people has become increasingly demanding on the work efficiency, intelligence and automation [1]. When the computer is widely used in people's lives, there is an increasing demand for all kinds of software needs. Writing large amounts of code makes software development inefficient and has high error rate. It is of great significance to study a new, fast and efficient system development method.

Since the emergence of model-driven architecture, it has greatly improved the efficiency of software development. It not only enhances the portability and interoperability between software, but also greatly improves the maintainability of the software. As the standard object-oriented modeling language, UML describes object-oriented systems from all angles. It corresponds to the most popular object-oriented programming languages. In the process of modeling of the object, the finite state machine clearly describes that the state of the object is how to convert and has implications to the outside world in the full life cycle. An important function of activity graphs is to describe algorithms and flows which can be used to refine the code framework. This paper focuses on code generation and presents a method to generate JAVA code directly through UML diagrams which can generate simple and easy to read code.

The basic concepts of UML in the code generation

Introduction to UML

UML (United Modeling language): unified modeling language is a universally applicable visual modeling language, well-defined and is easy to build document. In November 1997, all members of the object management organization took UML as the object-oriented modeling of the standard language. UML can provide static and dynamic view of a variety of software systems. Static views are class diagrams, object diagrams, use case diagrams, Components diagrams, and deployment diagrams. Dynamic views are state diagrams, activity diagrams, collaboration diagrams, and timing diagrams. In 2003, a new version of UML2.0 was introduced. UML2.0 provides an extension mechanism that allows the addition of new building blocks, the creation of new features and a mechanism for describing new semantics. UML models can be customized to suit specific themes and platforms. Prototype, tag value and constraint are three basic ways of UML extension mechanism.

Common Mechanisms of UML

Each language has its own flaws and UML model language can't describe all systems. In order to express UML graphics elements of the basic model can't show the information, UML provides a number of public mechanisms. Commonly used common mechanisms include conventions, modifiers, and extension mechanisms [2].

(1) Statute. In UML, a specification can be created after each element model representation. The syntax and semantics of the element are usually further presented in the form of textual descriptions [3].

(2) Modifier. Some of the basic model elements of the UML can be decorated with special symbols to express special meaning. For example, the graphic symbols of a class include class names, properties, operations, with abstract class names for italics and symbols for operations and the visibility of the property, such as "+" and "-".

(3) Expansion mechanism. It includes constraints, marker values and derivatives. Constraints are mainly used to extend the semantics of UML elements and can increase or modify the original rules. Marker values are mainly used to extend the characteristics of UML elements. Model elements can be given new information. Derivatives are mainly used to expand UML vocabulary and can create new model elements for different specific technology platforms.

Graphics of UML

Sate diagram

State diagram are used to describe the behavioral characteristics of the system. It describes all possible states of an instance and the transition between states by state, event, and conversion elements. It emphasizes the conversion of an object from state to another. The basic elements of the state diagram are: status, events, transformations and actions. The following elements are defined in the state diagram:

(1)State is a condition or situation that satisfies certain conditions，performs certain activities or waits for certain events in the object lifecycle.

(2)Migration. A state transition represents a direct relationship from the source state to the target state. State transitions are used to connect two states. The state indicated by the end of the arrow indicates the source state. The arrow indicates the target state. When a migration occurs, the state that the migration enters is called the active state. When the migration leaves a state, the state become inactive.

The syntax of the transition is: Event [Guardianship Condition] / Action.

An event is a description of an observable situation. A guard condition is a condition that must be met to trigger the transition and usually an expression of a Boolean type. When the corresponding event is triggered, if the expression value is true, the action will be executed and the migration is triggered. An action is a set of executable statements or calculations. Actions are atomic and uninterrupted. Therefore, the above migration syntax expression can be understood as: when the event occurs, the guardianship conditions and actions will occur. Each state in the UML state diagram has optional entry and exit actions. The enter action is executed when entering the state. The exit action is executed when exiting the state. The entry and exit actions are state-independent and independent of conversions. Regardless of how the state enters and exits, all its incoming and outgoing actions are executed.

(3)Event handling mechanism. The UML state machine's event handling mechanism is based on running to completion. The state machine's handling of an event is a "run to completion" [4]. In the RTC model, the system processes each event in a discrete and indivisible RTC step. High-priority time can't be interrupted to deal with other events. In the event processing, the system does not respond to other, so the transition between different states is not interrupted. The RTC step is a transition between the two state patterns of the state machine. The run-to-completion condition is set to avoid event handling conflicts in the concurrency state, so that the state machine can safely perform its RTC step and simplify the state machine migration semantics.

activity diagram

Activity diagrams are similar to flow charts and are similar to state diagrams. But its status indicates the action to be performed. The state of the system, concurrency state, decision points and other elements describe the control flow in the system. Activity diagrams can describe the work performed during the execution of an operation. It can also model the use case's workflow to show the path between the use case and the use case. It illustrates how an instance of a use case performs

actions and how to change the state of an object. It can also show the process of executing a set of related actions and how these actions affect other related objects. Activity diagrams can be used to identify how business operations are performed and the changes that may occur.

In summary, activity diagram models have the ability to describe system workflows and parallel activities. It makes the activity diagram model an important basis for system testing. In addition, the UML activity diagram model can be modeled at different levels of the system from system level, subsystem level to class level. So the activity diagram model can also guide the different levels of testing which includes system-level functional testing, integration testing and object-type unit testing.

UML language to object-oriented language mapping

The role of each graph in the software system is different. They describe a complete system from a different perspective. If the entire system as a whole by the code, the system can also be used to describe the graphical language. At present, many UML modeling software can be generated by the class diagram of the static structure of the system framework and can't generate executable code. To generate more complete code, we need to find a more complete mapping from the graphics to the code.

UML graphics to code ideas

Figure 1 is a simple state example.

The information that the state diagram can extract is shown in Table 1

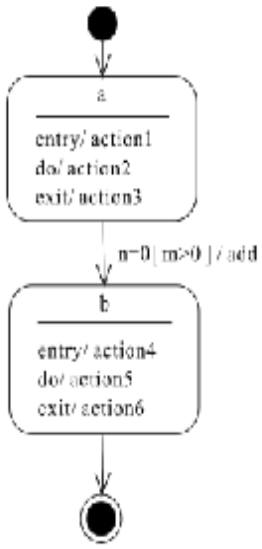

Figure 1. Example of a simple state

Table 1 State diagram extraction information

Information Type	Corresponding examples in the figure
State name	a
Belong to the class	easy
Entry action	action1
State internal activity	action2
Export action	action3
Is the next target state of the initial state	true
Whether the next state is the end state	false
Number of events	1
Migration event	n = 0
Migration action	add
Guardianship condition	m> 0
The name of the next state	b

Get the corresponding information code is as follows:

```
class state {
        static String current States ;
        int top =0;//Top of the stack
        state（String state）{
            current States = state ;
            top = 0;}
         void entry(){action1();}
          void do Sth(){action2() ;}
          void exit(){action3();}
          void action1() { }
```

```
void action2() { }
void action3() { }
Boolean infirst ;
Boolean event[];
Boolean guardcondition[];
Int eventcount ;
char eventaction[][];
char nextstate[];}
```

Generated code is as follows:

```
easy m = new easy() ;
m.action1();
m.action2();
while（next == true）{
if (n ==0) {
if (m >0) {
add();}
}}
m.action3();
```

Activity diagram

The corresponding translation code is as follows:

```
if（x1== x2）{
Avtivity2 ;
}else {
Activity3 ;
}
```

Figures 2 and 3 show different types of active states.

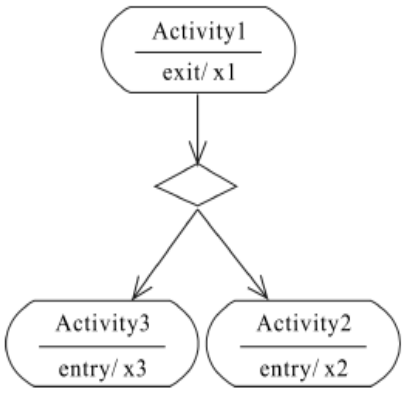

Figure 2. Active state with branches

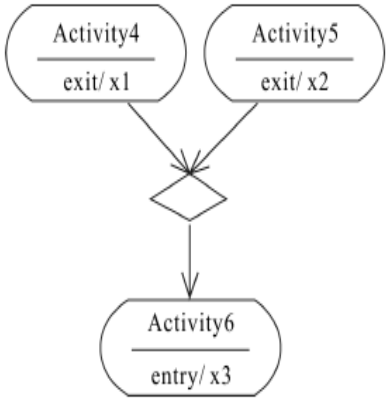

Figure 3. Active state with fusion

Corresponding code is as follows:

```
if （X1 | | X2）{
Activity6 ;
}
```

The translation code looks like this:

```
public class Thread1 {
```

```
public void run（String[] args）{
        Activity7 ;
        new Thread1（ " Thread1 " ）;
        new Thread2（ " Thread2 " ）;
        Activity10 ;
        }
    }
```

Figure 4 is the active state with concurrent activity.

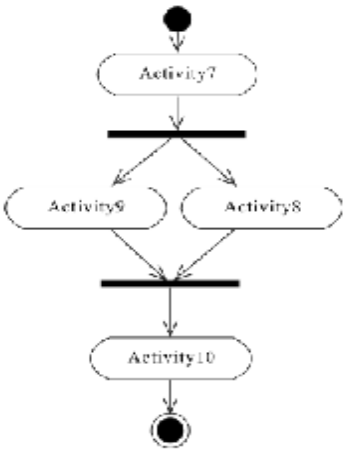

Figure 4. Activity status with concurrent activity

Examples of push stacks

A stack object has three states:
S1- - stack empty;
S2- - stack is not empty nor full;
S3- - Stack full.
The conditions for each state are as follows:

S1: When the object is initialized, it is done by the system. The empty operation set Empty () is executed in S2 state. The empty operation se-t Empty () is executed in the S3 state. Pop operation Pop () is executed in S2 state.

S2: Executes the push operation Push () in S1 state. Pop operation is executed in S3 state.
S3: Carry out the push operation Push () in S2 state.
To simplify the problem, we assume that the stack stack capacity of 2, the stack element data type is an integer.

The class diagram is shown in Figure 5.

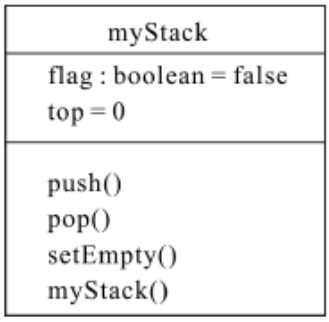

Figure 5. The stack to play the class diagram

Generated by the existing software code is as follows:

```
public class my Stack
 {
    private boolean flag  ＝false ;
    private int top  ＝0;
    public my Stack(){}
    public void push(){}
    public void pop(){}
    public void set Empty(){}
 }
```

The state diagram obtained from the above scenario is shown in Figure 6.

Pop () corresponds to the activities shown in Figure 7.

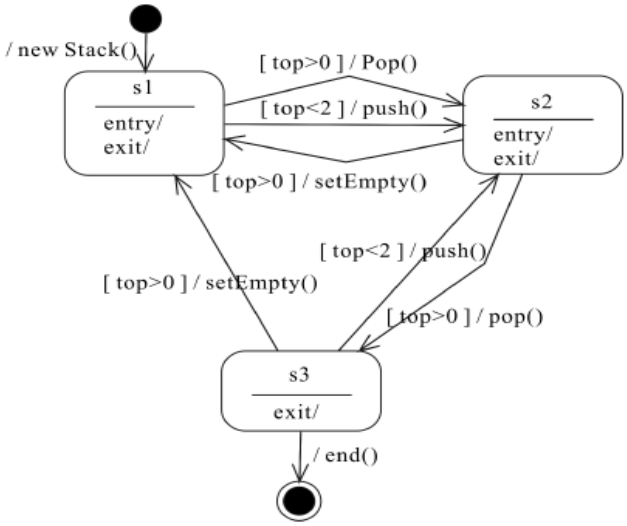

Figure 6. State of the activity in and out of the stack

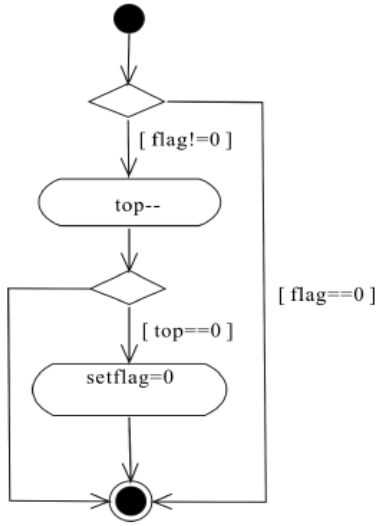

Figure 7. Pop () operation

The following code can be generated:

```
pop(){
  if(flag !=0){
     top-- ;
     if(top ==0){
     flag =0;
   } else{}}}
```

Push () corresponds to the activities shown in Figure 8.

Set Empty () corresponds to the activities shown in Figure 9.

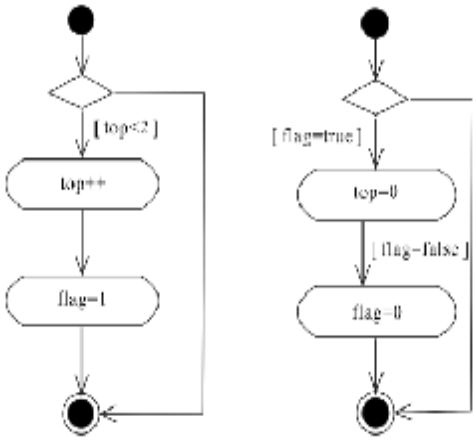

Figure8. Push() operation of the state.

Figure 9. Stack empty activities in the active state

In summary, we can use the existing class can generate generated executable code. The description of the convection in the activity diagram is the filling and description of the static structure of the class diagram. The state diagram describes the description of the object to complete the description of the dynamic behavior of the system and then it generates executable code. An example of pushing and playing the stack can generate code:

```
public class my Stack extends state {
    static final int TYPE__STACK__NULL =0 ;//Stack empty
    static final int TYPE__STACK__FULL =2 ;//Stack full
     static int stack Max =2 ;
     //Stack capacity maximum
     int top =0 ;//Top of the stack
     my Stack（String state）{
    super（state）; }
       private boolean flag ;
     public void my Stack（）{
       flag = false ;  }
     public void push（）{
```

```
if（top ＜ stack Max）{
top ＋＋ ;
System.out.println（ " " ）;
flag ＝ true ;
} else {
System.out.println（ " " ） ;}}
public void pop（）{
if（flag）{
top －－ ;
System .out.println（ " " ）;
if（top ＝＝0）{
flag ＝ false ; }
} else {
System .out .println（）;}}
public void set Empty （）{
if（flag ＝＝ true）{
top ＝0 ;
System.out .println（） ;
flag ＝ false ;
} else {}}
public String judge State（）{
if（top ＝＝ TYPE__STACK__NULL）{
current States ＝ " s1 " ;
return " " ;}
if（top ＝＝ stack Max）{
current States ＝ " s2 " ;
return " " ;
} else {
current States ＝ " s3 " ;
return " " ;}
}
public static void main（String args [ ]） throws IOException {
my Stack s ＝ new my Stack（current States）;
String correntstate＝ s ． judge State（）;
while（event&&guardcondition）{
eventaction;}
}
}
```

Conclusion

From the application point of view, code generation is mainly used in the following three areas: ① compiler and optimization areas. This paper mainly focuses on how to generate machine-generated code based on high-level language and the portability of generated code by using template matching code generation method, explanatory code generation method and driver code generation method. ② Generic code generation area. There are just generate the framework code of the software. Developers can choose the appropriate framework to describe the UML model into code. But the code is only a framework, and no specific implementation. This requires the developer to manually fill in the implementation code to meet the system functional requirements. ③ special code generation areas. Creating a small system or program for a dedicated domain can achieve a higher code generation rate.

References

[1] Leilei Kong. Unified Modeling Language UML [M]. Beijing: Tsinghua University Press, 2014.

[2] Dong-yang Guo. UML state diagram code framework based on template technology [D]. Xi'an: Xi'an University of Electronic Science and Technology, 2013.

[3] HAREL D. State charts: a visual formalism for complex systems [J]. Science of Computer Programming, 1987 (8), 231-274.

[4] Jinyu Song, Wei Yang Su. Research on model information automatic extraction based on UML state diagram [J]. Computer Engineering and Design, 2007 (20): 4860-4861.

[5]Jibing Ji. Feel the beauty of code generation [J]. China Modern Education Equipment, 2011 (6): 60-64.