Computer Life (CPL) ISSN: 1819-4818
DELIVERING QUALITY SCIENCE TO THE WORLD

Research on Automatic Generation Method of Java Code Based on UML Sequence Diagram

Shugang Liu[1, a], Jiajin Li[2, b]

[1]School of Computer Science, NORTH CHINA ELECTRIC POWER University, Baoding 071003, China;

[2]School of Computer Science, NORTH CHINA ELECTRIC POWER University, Baoding 071003, China.

[a]lsg69@qq.com, [b]972526556@qq.com

Abstract: Unified Modeling Language (UML) is a standard modeling language that analyzes and designs systems graphically. UML modeling can clearly represent the structure and behavior of the system. The class diagram in the UML model shows the static structure of each class in the system, and the timing diagram describes the time sequence of message passing between objects. This paper describes the research and implementation of the automatic generation method of Java code based on UML sequence diagram. In this paper, UML graphics are converted into XMI documents by using the eclipse plug-in trufun-plato. The corresponding metadata is extracted from the XMI document, combined with the proposed UML sequence diagram conversion rules, and the java code is generated by using the freemarker template technology.
Keywords: UML, Automatic code generation, Java.

---

1. INTRODUCTION

In the process of software development, developers always write some simple code, and every time new technology comes, they have to repeat the work over and over again. At the same time, changes in demand have never stopped. This has led to the following problems in program development:

(1). The code repetition rate is high. In the process of program development, programmers often do repetitive work. The most common is to access the database. Programmers should often write operations such as adding, deleting, changing, and paging. In order to avoid this problem, save a lot of mechanical entry time and duplication of work, improve work efficiency, and focus on the development of core business logic. A code generator for your own use is very important.

(2). The project is overdue. More and more pressure makes a software project, whether it is the end user, the enterprise, and the development team, want to complete it in the shortest time. What is counterproductive is that the time delay of the software project is widespread. Some surveys show

that 70% of the projects are beyond Estimated time. Large projects averaged 20% to 50% of the planned delivery time. The larger the project, the longer it takes to exceed the plan. The problem of speed of development has always been a top priority in the software development industry.

(3). Poor maintainability. In the development process of the project, once the customer proposes changes, it will make the project take the lead. Repeated code changes will slow down the project's infrastructure and code quality, making the code harder to understand and more robust.

In order to solve these problems, people have proposed automatic code generation technology. The significance of using code auto-generation technology is:

(1). Avoid repetitive work, automatic code generation reduces the writing of duplicate code, it can automatically generate a lot of repetitive code, which shortens the development cycle and improves code reuse rate.

(2). The consistency of the code style is good, the quality of the code directly affects the realization of the system function, and the good code generator is the result of the experience accumulation, the generated code is more robust, and the generated code defines the variable name, API calls and so on are very standard, which improves the readability and maintainability of the code.

(3). System design becomes dominant. In this way, we can spend more time on the design of the system business logic, thus greatly improving the software development efficiency.

(4). Quickly generate prototypes. According to the requirements, the user's functions can be quickly realized, and the prototype system can be constructed and developed on the prototype system.

(5). Easy to implement in multiple languages. The code generator uses a definition file that is independent of the programming language to express the application logic. Software engineers can convert the application logic into other programming languages or support programs on other platforms by creating template files, which is much easier than converting handwritten code. Much more.

This paper proposes a method based on UML time series transformation to java code. Starting from the UML metamodel, the transformation rules are proposed, and the XMI document is combined to generate java code.

2. CODE GENERATION PROCESS AND METHOD

Fig. 1 depicts the process framework for converting from UML models to Java code, including UML models, XMI documents, transformation rules, and java code. The UML model includes grammatically correct class diagrams and sequence diagrams that describe the static structure of the software system and business logic information, drawn by the programmer. The UML metamodel includes a meta-model of the class diagram and the sequence diagram, which is used to define the drawing rules of the class diagram and the sequence diagram. The code conversion rules are based on the characteristics of the UML model elements and the code structure of the Java language. When generating code automatically, first input the UML model that conforms to the UML metamodel rules, and then generate java code according to the transformation rules of the metamodel. The class diagram generates classes in java, and the specific implementation inside the sequence diagram generation method.

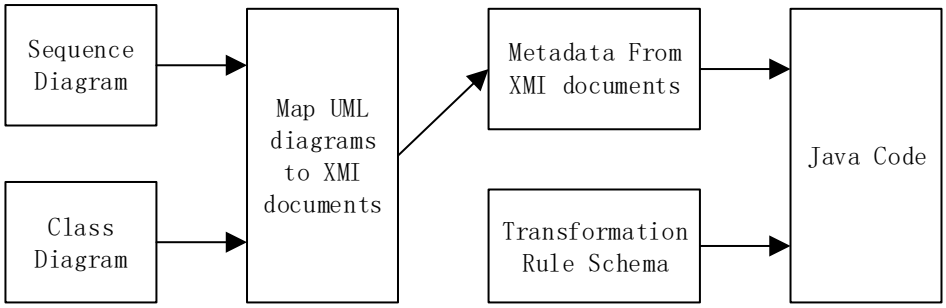


Fig. 1 Method of converting UML diagram to Java code

2.1 Metamodel of class diagram

The metamodel of UML defines the complete grammar rules for describing object models using UML, and Figure 2 is the metamodel for class diagrams. The classes in the UML class diagram correspond to Classes in the metamodel, and the attributes and operations correspond to Attribute and Operation. The parameter of the operation corresponds to Parameter, where kind indicates the type of the parameter. If kind=in indicates that the parameter is the parameter passed when the operation is invoked, kind=out indicates that the parameter is the return value of the operation. The association between classes is represented by Association. The type indicates whether the association type is one-to-one or one-to-many, and name is used to indicate the name of the class associated with it.

2.2 Metamodel of sequence diagram

Figure 3 is a metamodel of the sequence diagram. A sequence diagram is used to describe a method in a class that corresponds to Action and Operation in the metamodel. Where Operation is used to record the name and visibility of the method described by the sequence diagram. The object in the sequence diagram corresponds to the Object in the metamodel. The message communicated between different objects corresponds to the Message. The action to be executed in the message corresponds to the Action. The condition for the action is represented by recurrence, and the content of the action is represented by Request. Actions are divided into Call Action, Create Action, Send Action, Return Action, and Destroy Action. The return value of the invocation and creation operations is represented by Return Var. The type of the parameter is recorded in the Classifier.

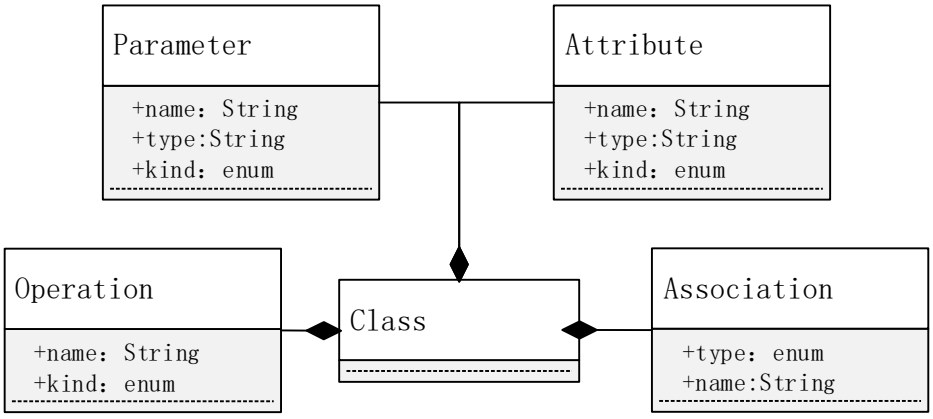


Fig 2.Metamodel of class diagram

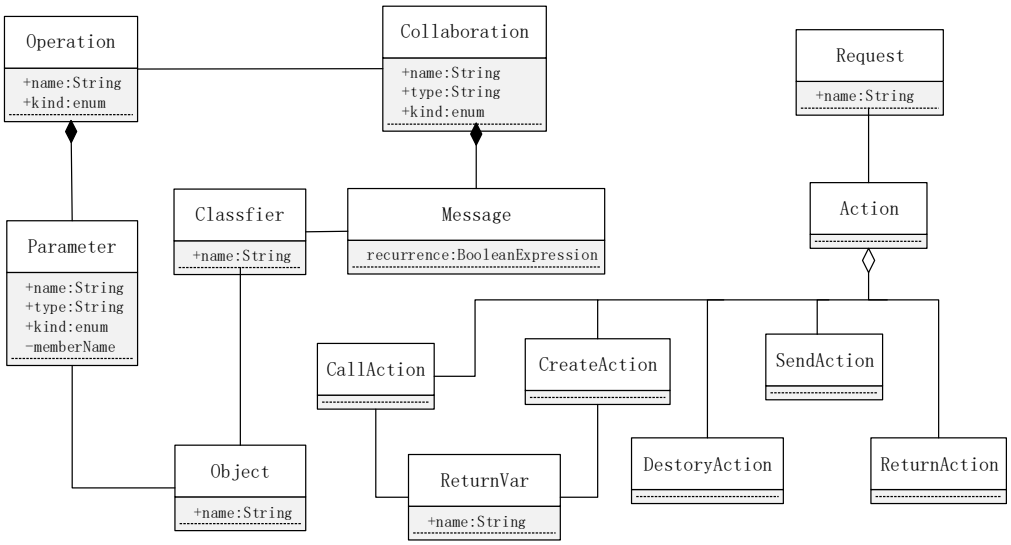


Fig 3.Metamodel of sequence diagram

3. CONVERSION RULES

In order to describe the conversion rules clearly and accurately, the conversion rules use a table description, in which the first column is the model element to be converted, the second column is the meta model corresponding to the model element, and the third column is the conversion rule corresponding to the meta model. Use uppercase letters to indicate non-terminal symbols, indicating that they can be replaced by the rules of response. Italic lowercase letters can be replaced by data in UML. All others are terminal symbols, which are the parts directly generating code.

3.1 Class diagram conversion rules

According to the correspondence between the UML model and the Java code, the relationship between the class diagram and the Java code is as follows: The classes, attributes, and operations in the UML model correspond to the classes, member variables, and member functions in java respectively. Therefore, the conversion rules for UML class diagrams to Java code include conversion rules for classes, attributes (including associations), and conversion rules for operations.

Rule 1. Class conversion rules. As shown in Table 1. The attribute name of the object c in the metamodel records the name of the class. In the conversion, only the name of the class in the UML model needs to be the class name of the generated java class.

Table 1. Class conversion rules

model element	metamodel	conversion rules

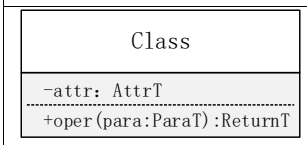
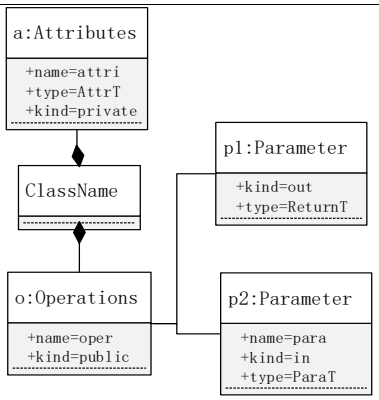
Class	c:Classifier
	+name=ClassName
		class c.name{
		ATTRIBUTES;
		OPERATIONS;
		}

Rule 2. Conversion rules for attributes and operations. As shown in Table 2. To be converted is a class containing attributes and operations. Objects o and a in the metamodel record the names of operations and attributes, respectively, p1 and p2 are the parameters and return values of the operation, respectively. The first line of the conversion rule defines the conversion of ATTRIBUTE. The

procedure, a.kind&a.type&a.name respectively represents the permissions, type and name of the attribute, and the combination of the three becomes the attribute definition in java. Use the symbol "_" to indicate the space in the code directly generated. OPERATION conversion mainly includes method permission, return value, name and parameter name, o.kind and o.name are used to indicate the type and name of the method, p1.type represents the return value of the method, PARAMS is the formal parameter of the method, and finally Instead of p2.type and p2.name, they represent the type and name of the formal parameter.

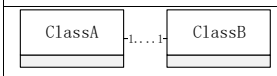
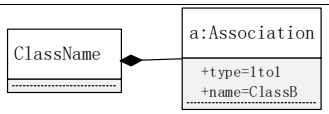
Table 2. Conversion rules for attributes and operations

model element	metamodel	conversion rules

		ATTRIBUTES->a..kind_a.type_a.name;
		ATTRIBUTES;
		OPERATIONS->o.kind_p1.type_o.name(PARAMS){
		SEQUENCES;
		}
		PARAMS->p2.type p2.name

Rule 3. Transition rule of association. As shown in Table 3. The association relationship in java is represented by the reference of the attribute. When it is one-to-one, you need to add your own reference to the attribute of the class. If it is one-to-many or many-to-many, you need to add a collection class to implement multiple mappings. GENERATE_NAME will be replaced by the user specification, or variable substitution will be automatically generated.

Table 3. Transition rule of association

model element	metamodel	conversion rules
		1 to 1->a.name GENERATE_NAME; 1 to many->Collection<a.name > GENERATE_NAME;

3.2 Sequence diagram conversion rules

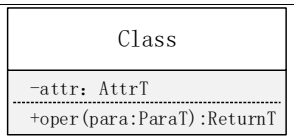
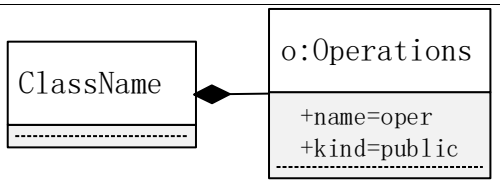
According to the characteristics of the UML model and the java code, the relationship between the sequence diagram and the java code is as follows: The content in the UML sequence diagram corresponds to the specific implementation details of the java member function, and the branches, function calls, and object creations in the sequence diagram correspond to java respectively. The if statement, the "." operator, and the new keyword. Therefore, the conversion rules of the sequence diagram to the java code include a conversion rule of the sequence diagram, a conversion rule of the condition, a conversion rule of the variable assignment, a conversion rule of the object creation, a

conversion rule of the method call, and a conversion rule of the message transmission. The message return and object destruction operations do not need to generate the corresponding code.

Rule 4. Sequence diagram conversion rules.As shown in Table 4. The object o indicates that the sequence diagram is a concrete implementation of the method oper( ). The non-terminal symbol SEQUENCE is replaced by LOCALDATA and MESSAGE, indicating that the implementation details of a method include the definition of local variables and the process of passing messages between objects.

Table 4. Sequence diagram conversion rules

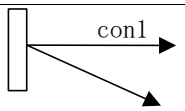
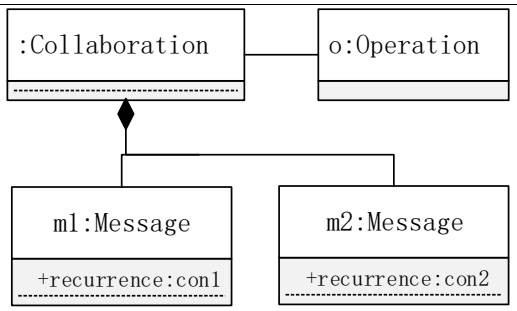
model element	metamodel	conversion rules

		SEQUENCE->{
		LOCALDATA;
		MESSAGE:
		}

Rule 5. Conditional conversion rule. As shown in Table 5. Object a sent a message containing the condition to object b. The attribute recurrence of the object m in the metamodel records the content of the condition, and when the condition is satisfied, the operation on the message is executed. Java uses an if statement to represent a condition, as shown in the third column of the table.

Table 5. Conditional conversion rule

model element	metamodel	conversion rules

		MESSAGE->
		if(m1.recurrence){
		MESSAGE_M1;
		}else if(m2.recurrence){
		MESSAGE_M2;
		}else{
		MESSAGE_OTHER;
		}

Rule 6. Conversion rules for variable assignment. As shown in Table 6. Assign the return value of the method oper( ) of Class B to the variable x. In the metamodel, the object rv represents the name of the variable, r records the name of the method, and c represents the return type of the function, that is, the type of the variable. The first line of the conversion rule shows the definition of the variable x. The non-terminal ASSIGNMENT in the second line indicates that the variable will be assigned. It can be assigned the return value of the function, as shown in rule 8, or it can be a constant or an expression. Assignment.

Rule 7. Conversion rules for object creation. As shown in Table 7. "<create>>" indicates that the message is an operation to create an object. In the metamodel, r represents the name of the object to be created, and p and c2 represent the name and type of the parameter. The first line of the conversion rule defines the parameters to be used when creating the object. The second line MESSAGE generates the code created by the object. In the third line, the non-terminal PARA is converted to the parameter

to be passed (PARAMETER contains the type and name of the parameter, PARA only Contains the name of the parameter).

Table 6. Conversion rules for variable assignment

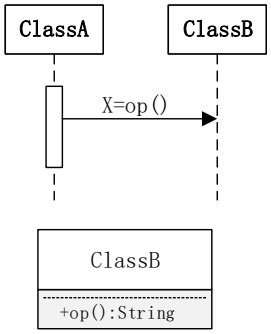
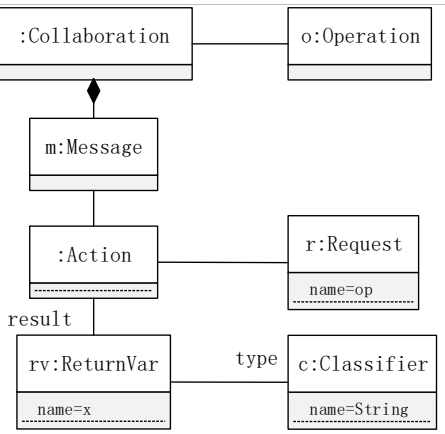
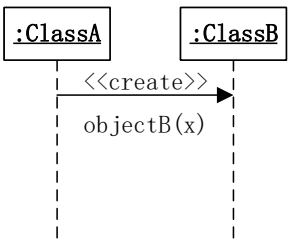
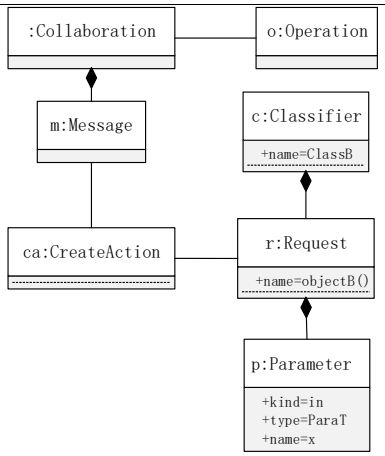
model element	metamodel	conversion rules
		LOCALDATA-> cname_rv.name; LOCALDATA; ASSIGNMENT-> rv.name=FUNCTION; FUNCTION->r.name;

Table 7. Conversion rules for object creation

model element	metamodel	conversion rules
		LOCALDATA-> c.name_r.name=MESSAGE;; MESSAGE-> new_c.name(PARA); PARA->p.name

Rule 8. Method conversion rules. As shown in Table 8. The content of the message is the method oper( ) in the object b of the calling object. In the metamodel, the name of the cr record object, r is the name of the method, and p and c2 are the names and types of the parameters. Method calls include calls to other object methods and calls to the object's own methods. Java uses the symbol "." to access member functions. The second line of the conversion rule shows the rules for calling methods of other objects and assigning the result to a variable. When calling its own method, remove cr.name. The conversion rules for methods that have no return value are shown in the fourth line of the table.

Rule 9. Conversion rules for message delivery. As shown in Table 9. The message does not contain any method calls or content created by the object. In the metamodel, r records the contents of the message. Messages are sent including messages sent to other objects and reflex messages sent to themselves. Both need to generate code directly from the content of the message, as shown in the third column of the table.

Table 8. Method conversion rules

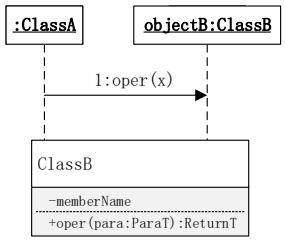
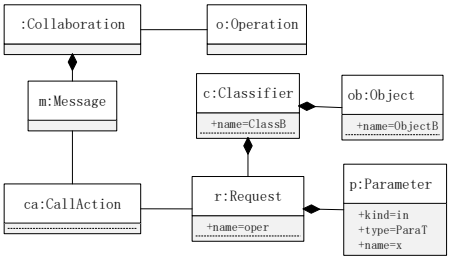
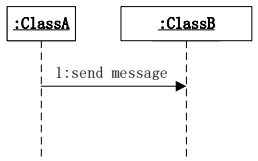
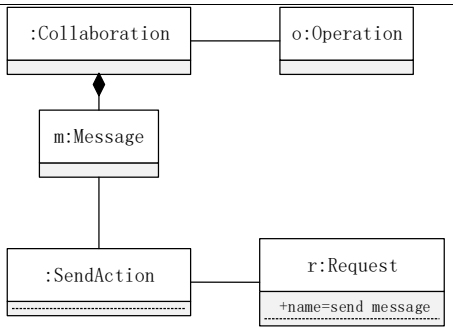
model element	metamodel	conversion rules
		LOCALVAR-> p.type_p.name; MESSAGE-> ob.name.r.name(PARA); PARA->p.name; MESSAGE-> ob.name.r.name(PARA);

Table 9. Conversion rules for message delivery

model element	metamodel	conversion rules
		MESSAGE->r.name;

4. EXPERIMENT AND RESULT ANALYSIS

This article takes the part of the student achievement management system as an example to describe how to use the class diagram and time series diagram to generate java code containing structure and behavior information. The class diagram is shown in Figure 4. There are two entity classes, Student and Grade, with annotations. The form indicates the constructor and the get/set method. UserInterface processes the data and sends the information to the service. The service calls Dao's corresponding method to complete the addition, deletion, and modification of the information.

Figure 5 is a sequence diagram corresponding to the update() method in UserInterface. The user interface first gives a prompt for inputting information, and then the user inputs the corresponding sId, gId, gNum in the interface, and then creates the Student and Grade objects respectively, and calls setGrade by calling The () method sets the value of the Grade associated with the Student, and then calls service.update() to get the return value to determine if the insertion was successful.

The UserInterface generation process is as follows:

(1). Using rules 1, 2, 3, generate various methods and properties in the class diagram, and generate a framework for the update method. The internal implementation is implemented by a sequence diagram. Here, for convenience, annotations are used to represent the constructor and get/ Set method.

(2). Using rule 4, generate the internal framework of the update method.

(3). Using rule 9, generate prompt information;

(4). The second step is to simplify the original steps, here the system class Scanner is called, the input and assignment are implemented, and the rule 6 is generated to generate the code.
(5).Using rule 7, create a student and grade object.
(6). Using rule 8, call the setGrade() method of the Student class.
(7). Using rule 6, call the service update method and return the value to bool.
(8). Using rule 5, generate a conditional statement and give the user feedback information.
The resulting code is shown in Figure 6:

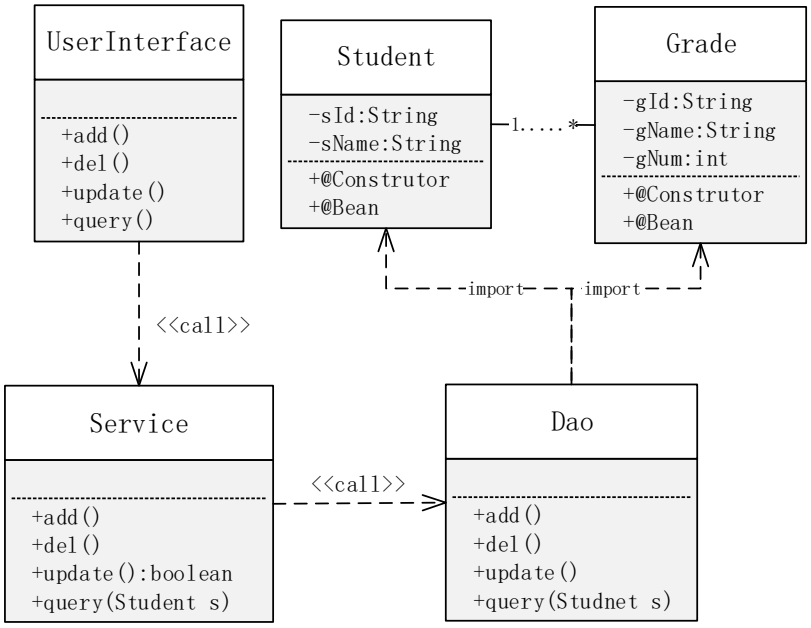


Fig 4.class diagram to student achievement management system

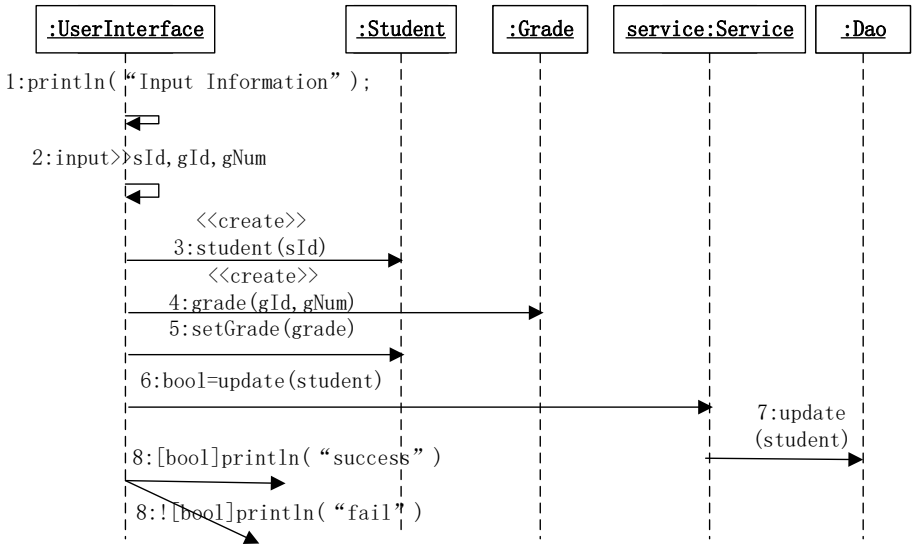


Fig 5. Sequence diagram to update() method

```
/**
*AUTOGENERATE
*/
class UserInterface{
    public void update(){
        println("Input Information");
        Scanner sc = new Scanner();
        String sId;
        sId=sc.next();
        String gId;
        gId=sc.next();
        int gNum;
        gNum=sc.nextInt();
        Student student = new Student(sId);
        Grade grade = new Grade(gId,gNum);
        student.setGrade(grade);
        Boolean bool = service.update(student);
        if(bool){
            println("success");
        }else{
            println("fail");
        }
    }
}
```

Fig 6. Conversion result

5. CONCLUSION AND FUTURE WORK

This paper combines the combination of class diagrams and timing diagrams and metamodels, proposes the rules for converting time series diagrams into java code, and converts UML diagrams into XMI documents through trufun plato plugin, XML data obtained by DOM4J technology parsing, and freemarker template. The combination generates the target code. Compared to methods that only generate frames. The code integrity of the proposed method is higher. However, there are still no rules for transforming elements such as fragments in the sequence diagram. For some common instructions, there is no uniform specification. Therefore, the next step is to improve the existing rules and propose conversion rules for the activity diagram.

REFERENCES

[1] Chen Hui. Research on formal methods of UML sequence diagram and state diagram [D]. Nanjing: Nanjing Normal University, 2008.

[2] Zhang Sen, Deng Lei, Wu Jian, et al. A Code Generation Method for Distributed Object Model Based on MDA_Zhang Sen[J]. Journal of Northwestern Polytechnical University, 2014, 32(1): 49-54.

[3] Dong Hyuk Park, Soo Dong Kim, "XML Rule Based Source Code Generator for UML Case Tool", Asia-Pacific Software Engineering Conference (APSEC2001), pp. 53-60, 2001.

[4] Wang Xiaoyu, Qian Hongbing. Research on automatic generation method of C_code based on UML class diagram and sequence diagram_Wang Xiaoyu[J]. Journal of Computer Applications and Software, 2013, 30(1): 190-195.

[5] Liu Ran, Chen Ying, Zhao Xiaolin. 99 Automatic Generation of Code Based on UML-based CASE Platform[J]. Journal of Beijing Institute of Technology, 2002, 22(2): 196-200.

[6] D. Kundu, D. Samanta, R. Mall, "Automatic code generation from unified modelling language sequence diagrams", Software IET, vol. 7, no. 1, pp. 12-28, 2013.

[7] Cui Meng, Yuan Hai, Shi Yaoxin, et al. 148 A MMA-based UML sequence diagram to state diagram conversion method_Cui Meng[J]. Journal of Nanjing University(Natural Science), 2004, 40(4) : 470-482.

[8] A. Parada, E. Siegert, and L. de Brisolara, "Generating java code from uml class and sequence diagrams," in Computing System Engineering (SBESC), 2011 Brazilian Symposium on, nov. 2011, pp. 99- 101.

[9] Su Hongjun, Yan Yunshan, You Zhenhua. Model-driven transformation from 126 UML model to N-layer Web model[J]. Journal of Computer Applications, 2014, 34(4): 1161-1164.

[10] B. Selic, "Uml 2: A model-driven development tool. modeldriven software development," IBM Systems Journal, Riverton, vol. 45, n. 3, pp. 607-620, 2006. (Pubitemid 44323029)