Computer Life
ISSN: 1819-4818 | Vol. 12, No. 2, 2024

Research and Implementation of Java Code Automatic Generator Based on UML

Shugang Liu, Xiaotian Gao

School Of Computer Science, NORTH CHINA ELECTRIC POWER University, Baoding 071003, China.

Abstract: As a model-based tool, automatic code generation technology has been known and used by more and more developers due to its efficiency and convenience. This article combines UML and automatic code generation technology, proposes the conversion rules between UML model diagrams and Java code, and uses object-oriented and modular design ideas to divide the Java code automatic generator into input layer, data verification layer, and code. Generation layer, each layer uses the form of a unified interface for data exchange. Designed and implemented a Java code automatic generator based on UML model diagram. The main contents of this paper are as follows:(1)Study the mapping rules for converting UML class diagrams and sequence diagrams into Java code, extract the elements and relationships of various model diagrams, build a metamodel of UML model diagrams, and study the mapping between metamodels and Java statement-level code Relationship, and put forward sentence-level code conversion rules.(2)Using DOM, Xpath and trufun-plato plugins to parse the metadata in UML model diagrams. Through the previous study of UML class diagrams, sequence diagrams and state diagrams to the mapping rules of Java code, the coding realized automatic generation of Java code based on UML And verified the proposed Java code mapping rules.Through tests and practical use proofs, the UML model-based automatic Java code generator can convert UML model diagrams into parts of Java code, verifying the correctness and feasibility of the rules studied.

Keywords: UML, Code generation, Java, meta model.

1. Introduction

1.1. Research background.

With the development of software development, software tends to be structured and modular. With the emergence of a large number of frameworks and the popularity of micro-services, software development has formed a development form in which convention is greater than configuration and configuration is greater than coding. Developers began to focus on the core algorithm and core business writing, however, a large number of repetitive code still occupied a large number of developers' time, reducing the development efficiency and delaying the project cycle. In addition, due to the deviation between Party A and Party B in software cognition and description, most of the software development in software engineering stays at the stage of requirements analysis and overall design. Although the development mode of agile development has been put forward at present, the frequent changes in requirements make the development still slow. Most of the complete code and architecture need to be pushed back and reconstructed, which makes the code basic framework obscure and difficult to understand, and the code quality is gradually reduced. The robustness and integrity of the system are more fragile. After the project expires, it can only go online in a hurry. The project maintenance is poor, which brings a lot of work to the project maintenance.

Based on the above problems, developers expect to have the technology of converting the model diagram generated by the requirements analysis, outline design and other stages into the corresponding code framework and code statements in a certain way. The developer encapsulates the core code business and embeds it into the corresponding code framework, and completes the project development on the basis of minimizing or not changing the code converted from the model. After that, the frequent changes of requirements only need to change the corresponding model diagram, generate the corresponding code framework, and reduce the parts modified by developers. Unified modeling language UML and automatic code generation technology are the best choices to solve the above problems.[1,2].

Object-oriented language began in the 1970s, and reached its peak in the 1980s and 1990s. Various modeling methods emerged. The creators of the language highly praised their own products.[3] UML combines the advantages of other languages and further develops on this basis to form the most commonly used modeling language. The unified modeling language UML constructs the system structure and some dynamic information into corresponding models through semi-formal modeling language.[4] Describe each package, component, class, and state in the system in the form of text, graphics, and symbols. It is applicable to the modeling of complex projects, can well describe problems, describe solutions, and provide a good way for both parties to communicate.

This paper takes UML model diagram and Java code automatic generation as the main research direction, combines the model end of UML modeling language and Java automatic code generation rules, and studies and implements the Java automatic code generator based on UML. Its main significance is:

(1)Automatic code generation avoids a lot of redundant code and redundant work, reduces the code duplication rate and improves the development efficiency.

(2)UML modeling language can well explain specific requirements to users, provide ideas for developers, and help developers more.

(3)The automatically generated code is more conducive to the encapsulation and maintenance of the code, and can realize the unified code writing style and the standard code writing format. It has good readability.

(4)Through automatic code generation, the model diagram modified by the requirement analysis can directly generate the corresponding code, making software modification easier and less costly.

1.2. Research status.

Many large software companies have examples of using automatic code generation tools. As an excellent UML modeling software, Rational Rose, launched by Rational, can not only draw all UML 2.0 model diagrams, but also provide code automatic generation plug-ins that map class diagrams to Java code. In addition, the Target Center launched by Target can also automatically generate Java object code through UML files.[5,6] At present, wizards in most Java compiler environments, such as eclipse for Spring and Intel HTIDEA, can automatically generate corresponding project frameworks and configuration files for users. And the built-in plug-ins can generate corresponding construction methods, get/set methods, and override toString methods according to Java class properties. At present, the mainstream Dao-layer database framework hibernate can automatically generate database files and SQL query statements related to the configuration database through the self-defined database independent statements HOL and Criteria. The database framework mybatis, which is also the Dao-layer, can generate corresponding Java simple objects (POJO) and Java query objects (POJO RESULT) as well as corresponding mapper files and basic addition, deletion and modification methods through database files.[7,8]

In general, foreign research on automatic code generation mainly includes the following aspects:

(1)The code is automatically generated by corresponding design patterns. The common design patterns mainly include singleton, factory, static factory and other patterns. The structure of these patterns is relatively simple and clear. It is easy to generate specific code in terms of framework.[9]

(2)Web pages are automatically generated. For Java web, early Web pages wrote HTML statements through the servlet's write method. Most of the work of developers is to write repetitive write statements and splice strings. With the development of automatic code generation, jsp technology has emerged, enabling jsp templates to generate corresponding servlets, and then to the current template language freemarker, velocity. Greatly liberated developers.[10]

(3)The code generated by the relational model is mainly used to generate POJO and corresponding Java code automatically from the relational database, and generate database files and SOL statements from the POJO and corresponding Java code.

(4)Automatically generated based on UML model. Take UML model as input, map Java code through UML model, and propose UML mapping Java code relationship. It is mainly represented by the Model Driven Architecture (MDA) proposed by OMG.[11]

In addition, in order to promote the further development of code generation technology, more representative meetings include the International Symposium on Code Generation and Optimization (CGO) and the Code Generation Conference. The World Computer Society has achieved great success by holding a conference on code generation and optimization to study and discuss the research progress of code generation and optimization technology. The code generation meeting specifically focuses on model-driven software development and code automatic generation, including code release pool, model-driven framework, eclipse modeling tool, executable UML, etc.[12,13].

2. Principle of UML Model Code Automatic Generation

2.1. Code generation based on template parsing.

Code generation based on template parsing is mainly controlled through template files and data files as inputs to generate code. The commonly used Jsp technology and struts framework in Java web use this method for automatic code generation. This method first determines the static parts in the file, such as the position, layout, font, etc. of each part in the JSP. Then, the static parts are saved in a fixed format, and other dynamic parts are filled in the form of placeholders to form a JSP file. Finally, the data to be displayed is responded to the path page represented by the JSP in the form of attributes. The JSP dynamically fills in the data after receiving it, Finally, form the Servlet.class file to be displayed. Its main advantage is that the code generation concept is simple, and users can customize templates according to their needs, making it easy to achieve simple and large-scale code generation.

2.2. Code generation based on Relational model.

The code generation of the Relational model is mainly applied to the Dao layer relational database. The mybatisGenerator plug-in provided by the mybatis framework is based on the relational database to generate the POJOs and query classes of the corresponding database. The mybatisGenerator will convert the database name in the database to a class, convert the database column name to an attribute, and convert the mapping relationship of foreign keys to corresponding set attributes. MybatisGenerator first reads the database. SQL file to find the Relational model built by the corresponding table building clause, and then extracts key column names, primary and foreign keys and other data. Finally, fill the data into the pre written template file and generate the final Java code. In addition, the hibernate framework can also generate SQL statements through configuration files and program control, which is also a code generation method based on the Relational model. Relational model code generation still uses template parsing, but compared with template parsing, algebraic Relational model is more widely applicable. The advantage of Relational model code generation lies in its applicability and accuracy to Relational model, while its disadvantage lies in its inapplicability to non Relational model.

2.3. Model driven code generation.

Model-driven architecture (MDA) is a software development framework proposed by OMG in 2001 [14-16]. This framework focuses on the model and stores it in a standardized manner, using the model to accurately describe the entire system. Separating requirement analysis and specific design is beneficial for the development of large and complex systems. The main working principle of MDA is shown as follows:

(1) Requirements analysis establishes a Platform Independent Model (PIM).

(2) Using MAD tools, convert PIM into Platform Specific Model (PSM).

(3) Generate specific code based on PSM, and finally test the generated code.

This article mainly studies the automatic generation of code from SM to code. Overall, the above three code generation technologies all adopt template parsing, but overall, MDA's architecture is more complete and supportive. Therefore, this article mainly uses MDA to map and generate Java code.

2.4. Code automatic generation tool.

This article studies, designs, and implements a Java code autogenerator based on UML. Design and implement according to the concept of modularization. The various modules are injected through interfaces, and the specific selection techniques are as follows:

(1) Trufun-plato plugin

Trufun-plato is a visual modeling tool based on UML. This time, we used the Trufun-plato plugin developed based on the Eclipse plugin. Trufun-plato saves the directly drawn UML model diagram in XML format as a. trufun.tmx file, which is convenient for extracting metadata from the UML model diagram.

(2) DOM

This paper mainly uses the method of transforming UML model diagrams into XML documents to extract data. At present, the technologies of parsing and extracting relevant data from XML mainly include the Document Object Model and the Simple API for XML (SAX) [17].

DOM technology mainly focuses on loading XML documents into memory in a tree like structure, and then traversing and searching for relevant nodes according to the relationship between the parent and child nodes of the tree. And due to operating the DOM tree in memory, the query and extraction speed will be relatively fast.

SAX is mainly used to read large XML documents, supporting fast document reading without the need to load all document contents into memory. Due to the small size of the XML document converted in this article, we need to frequently search for the corresponding metadata, so we use DOM for parsing.

For the Java language, dom4j.rog has produced the XML parsing package DOM4J, which defines five basic objects in DOM: Document, Node, NodeList, Element, and Attr. Document corresponds to the entire starting point of the XML document, equivalent to the root node of the DOM tree. The Node object is the most basic object in the DOM structure, representing an abstract node in the document tree. However, node objects such as Element, Attr, and Text are often used to manipulate and store data. The Element object represents the tag element in an XML document, inherited from Node, and can contain attributes or store corresponding data in the tag. The Attr object represents the attributes of a certain label. Attr also inherits from Node, but is often used as part of the Element object and not as a separate node in the DOM tree.

3. Implementation of Java Code Autogenerator Based on UML

3.1. requirement analysis.

Requirement analysis refers to a detailed analysis of the design of Java code automatic generator for the problem to be solved, clarifying the requirements, types of input data, content, and final results. Previously, people believed that requirement analysis was the simplest step, but as software grew in scale and had more functions, requirement analysis, as a part of system design, was even more important than system development. For this system, we will conduct a simple analysis as follows:

(1) Function: Complete the automatic generation of class diagrams, sequence diagrams, and state diagrams that comply with UML specifications into executable code in Java language.

(2) Input: Convert the UML model diagram into an XML document through the trufun plato plugin.

(3) Output: executable Java Language code.

3.1.1. Overall design.

The functional descriptions of each component are as follows:

(1) Model Data Parser: Interprets system state machine model data, including user built PIM, PSM, and various meta models.

(2) Basic Toolset: Set up recognizable metamodels in transformations.

(3) Code generation template: The template implementation of model to code conversion rules. In this article, the conversion rules from the state machine model to the Java language are the core content of the code generator.

(4) Configuration file: An auxiliary tool used in the configuration code generation process.

Workflow of Code Generator: The code generator first reads the configuration file, obtains the location of the UML model, loads the UML model, and then parses the corresponding UML metadata through the model data parser. The UML metamodel and Java language mapping rules will be read into the program, and then the metadata in UML will be filled into the corresponding Java language template file, which will be controlled by the template processing engine and output the corresponding Java Language code file.

3.1.2. Overall implementation.

The Java code generator involved in this article runs on the Microsoft Windows 11 operating system, using Java language and Freemarker template language. The development tools are IntelliJ IDEA 2020 and JDK 1.8. The Trufun plat plugin from eclipse is used for modeling, and the Spring and Maven frameworks are used to manage system resources.

Explanation of each structure:

(1) Model data parser: Use DOM4J and Xpath to parse data.

(2) Basic toolset: UML 2 metamodel, custom rule set.

(3) Code generation template: Freemarker template file customized according to Freemarker template language.

(4) Configuration file: Custom configuration file.

The Java code automatic generator divides the process of generating Java code from a UML model into four parts:

(1) Parsing the PSM model through the trufun palto plugin to obtain an XML form of the UML element model.

(2) Parse the UML element model in XML form using DOM4J and xpath to obtain metadata.

(3) Define mapping rules and encode the mapping relationship between mapping rules and the metamodel.

(4) Run the freemarker template, fill the metadata into the template, and generate the corresponding Java code.

3.2. Implementation of Java Code Autogenerator Based on UML.

3.2.1. Basic configuration implementation.

The basic configuration of Java code generator is used to ensure the applicability of code generator parsing, including data model parser file configuration and template file configuration.

(1) Parser File Configuration

Java code generator input model is an XML format file that conforms to the UML 2.0 standard and is stored through the trufun plato plugin. When the Java code autogenerator needs to parse the model, the model data parser reads this XML file. For Java, we mainly use the DOM 4J method to parse this XML file and obtain the corresponding relevant data through xpath.

(2) Template Configuration File

In order to facilitate users' use of the code autogenerator and enhance the robustness and usability of the Java code autogenerator, corresponding configuration files are provided for users to choose from.

The following explanations are provided for the configuration file information:

(1) File configuration format: used to configure the encoding format of the generated Java code, which is a required item and can also be set to other .

encoding methods that comply with international standards to avoid garbled code problems during the code generation process.

(2) Template processing engine: As the template file implemented in this article is implemented using the Freemarker template engine, the template engine uses the Freemarker template engine for parsing.

(3) Model parsing tool: This article uses the trufun plato plugin to convert the UML model into the corresponding XML file, so the parsing tool uses DOM4J. The configuration here and the executable file format introduced below are used accordingly.

(4) Code Generation Address: Used to specify the file location after code generation. If not specified, it will be in the current directory.

(5) Processable file format: corresponding to the model parsing tool mentioned earlier.

(6) Code Generation Directory: Used to specify the directory structure after code generation, which is the package name in Java code.

(7) Template file information: used to specify the file location and name of the UML model diagram stored in XML form by trufun plato.

(8) Exception handling information: A reserved item used to specify the handling method when an exception occurs during code conversion. Currently empty.

(9) Abnormal filling characters: What kind of characters are used to fill in the parts that cannot be parsed by the template, and the developer will supplement them later. Currently reserved.

3.2.2. Implementation of Java Code Autogenerator Template.

The Java code automatic generator template needs to combine rules to fill the required metadata into the corresponding template. In addition, considering the correlation between models, we adopt the method of introducing other templates to explain the freemarker template file as follows.

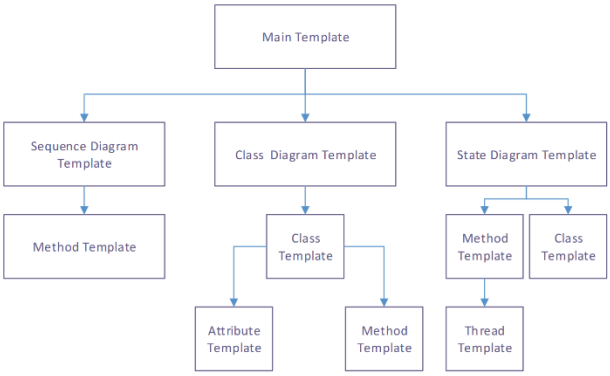

Fig. 1 Freemarker template file structure diagram

Fig 1 show the Freemarker template file diagram, with the Main Template serving as the entry point for the entire Freemarker template file. Other template files are loaded into the Main Template through import. They are class diagram templates, sequence diagram templates, and state diagram templates. Class diagrams are converted into classes, which contain both attributes and methods. Sequence diagrams are converted into specific implementations of corresponding methods. State diagram conversion to class and method does not contain attributes, and in addition, the transition design in the concurrency semantics of the state diagram is designed to be multi-threaded, so it is also necessary to introduce a thread template.

3.3. Running and Testing of Java Code Autogenerator Based on UML.

This test runs on the Windows 10 operating system, with the input data being a class diagram, sequence diagram, and state diagram stored in the trufun plato plugin that complies with the UML 2.0 standard. Draw class diagrams, sequence diagrams, and state diagrams that comply with the UML 2.0 standard in eclipse with the trufun plato plugin. The trufun plato plugin stores graphics as XML standard format files with a suffix of. tmx.

This article takes the part of the student management system as an example to describe how to use class diagrams and sequence diagrams to generate Java code containing structural and behavioral information. The class diagram is shown in Fig 2, which includes two entity classes: Student and Grade. The construction method and get/set method are indicated in the form of annotations. The UserInterface processes data and sends information to the service, which calls the corresponding method of Dao to complete the addition, deletion, modification, and query of information.

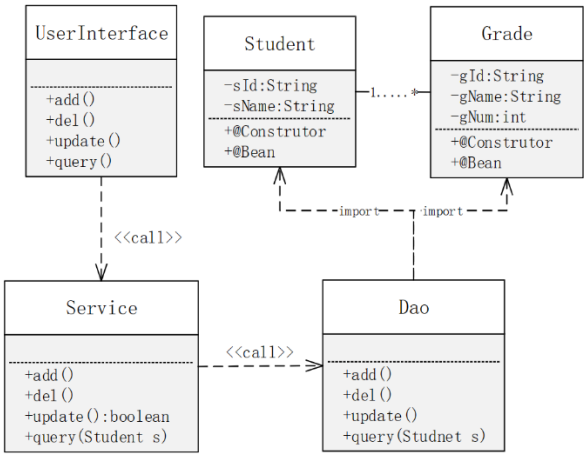

Fig. 2 Class diagram of student management system

Fig 3 is the sequence diagram corresponding to the update() method in the UserInterface. The user interface first gives a prompt for entering information, and then the user enters the corresponding sId, gId, gNum in the interface. After that, the student and grade objects are created respectively, and the value of the Grade associated with the student is set by calling the setGrade() method, and then the return value is obtained by calling service. update() to judge whether the insertion is successful.

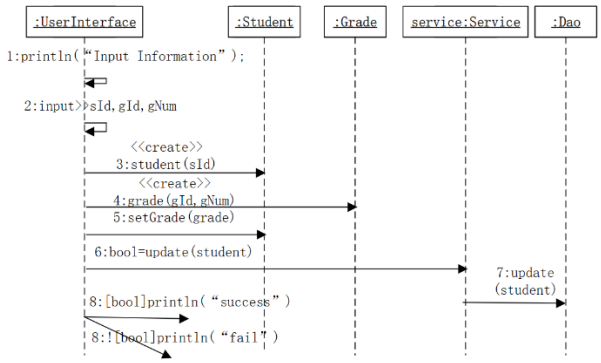

Fig. 3 Update() method sequence diagram

After comparing the model with the transformation rules, the UML based Java code automatic generator meets the statement level code generation requirements. Through testing, it is verified that the generated code is accurate and the code running results are correct. In the backend code of the student management system, the effective code ratio for class diagram and sequence diagram conversion is over 85%, and the effective code ratio for state diagram conversion is over 50%, achieving the expected goal.

4. Summary

This article studies and implements the related technologies of Java code automatic generator based on UML model diagrams. This mainly includes the study of the principle of code automatic generation, class diagrams and sequence diagrams, as well as the mapping relationship between state diagrams and Java code. In addition, the design and implementation of Java code automatic generator have been carried out. The expected conversion between UML model diagrams and Java code has been basically completed. According to the mapping rules obtained by the research institute, a Java code autochanger was coded and implemented. Overall, compared to methods that can only generate frameworks, the code generation method proposed in this article has relatively higher integrity.

This article mainly studies the conversion between static model class diagrams and dynamic model sequence diagrams, as well as between state diagrams and Java code, in order to achieve partial metadata to statement level conversion. Although code mapping may not necessarily require all metadata, there are still some metadata that can be mapped to statement level Java code, which has not been researched yet. In addition, this article uses a metamodel to standardize the sequence diagram and state diagram, so that they represent different system codes. For the same system, there is a certain correlation between the state and sequence diagrams, which can be transformed or complementary to each other. Later, the correlation between the state and sequence diagrams can also be studied to make the generated code more complete.

References

[1] Quan Long, Zhiming Liu, Xiaoshan Li and He Jifeng, "Consistent code generation from UML models," 2005 Australian Software Engineering Conference, Brisbane, Queensland, Australia, 2005, p. 23-30.

[2] F. Ciccozzi, A. Cicchetti and M. Sjödin, "Towards Translational Execution of Action Language for Foundational UML," 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, Santander, 2013, p. 153-160.

[3] S. E. V. and P. Samuel, "Automatic Code Generation From UML State Chart Diagrams," in IEEE Access, vol. 7, p. 8591-8608, 2019.

[4] M. Thongmak and P. Muenchaisri, "Design of rules for transforming UML sequence diagrams into Java code," Ninth Asia-Pacific Software Engineering Conference, 2002., Gold Coast, Queensland, Australia, 2002, p. 485-494.

[5] D. Torre, Y. Labiche, M. Genero, M. T. Baldassarre and M. Elaasar, "UML Diagram Synthesis Techniques: A Systematic Mapping Study," 2018 IEEE/ACM 10th International Workshop on Modelling in Software Engineering （MiSE）, Gothenburg, Sweden, 2018, p. 33-40.

[6] M. Usman, A. Nadeem and T. Kim, "UJECTOR: A Tool for Executable Code Generation from UML Models," 2008 Advanced Software Engineering and Its Applications, Hainan Island, 2008, p. 165-170.

[7] N. S. Bhullar, B. Chhabra and A. Verma, "Exploration of UML diagrams based code generation methods," 2016 International Conference on Inventive Computation Technologies（ICICT）, Coimbatore, 2016, p. 1-6.

[8] G. A. Papadopoulos, "Automatic code generation: A practical approach," ITI 2008 - 30th International Conference on Information Technology Interfaces, Dubrovnik, 2008, p. 861-866.

[9] S. E. Viswanathan and P. Samuel, "Automatic code generation using unified modeling language activity and sequence models," in IET Software, vol. 10, no. 6, p. 164-172, 12 2016.

[10] A. Tarasiev, M. Filippova, K. Aksyonov and O. Aksyonova, "Developing Prototype of CASE-Tool to Create Automation Systems Based on Web Applications Using Code Generation," 2018 Dynamics of Systems, Mechanisms and Machines （Dynamics）, Omsk, 2018, p. 1-4.

[11] M. K. Shiferaw and A. K. Jena, "Code Generator for Model-Driven Software Development Using UML Models," 2018 Second International Conference on Electronics, Communication and Aerospace Technology （ICECA）, Coimbatore, 2018, p. 1671-1678.

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

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

[14] Sen Zhang, Lei Deng, Jian Wu, et al. Code Generation Method of Distributed Object Model Framework Based on MDA[J]. Journal of Northwestern Polytechnical University, 2014, 32（1）,p. 49-54.

[15] Xiang Chen, Xuebin Wang, Quanyuan Wu. Implementation of Code Generation Technology in MDA[J]. Computer Applied Research, 2006（01）,p. 147-150.

[16] H. Benouda, M. Azizi, M. Moussaoui and R. Esbai, "Automatic code generation within MDA approach for cross-platform mobiles apps," 2017 First International Conference on Embedded & Distributed Systems （EDiS）, Oran, 2017, p. 1-5.

[17] Qiong Zhang, Pian Huang. XML-based Code Generating Tool[J]. Electronic technology, 2015, 28（02）,p. 95-97.