A Method of Applying Component-Based Software Technologies to Model Driven Development

Keinosuke Matsumoto, Tomoki Mizuno, and Naoki Mori
Dept. of Computer Science and Intelligent Systems
Graduate School of Engineering, Osaka Prefecture University
Sakai, Osaka, Japan
matsu@cs.osakafu-u.ac.jp

Abstract—**Improving the efficiency of automatic software generation is a problem of model driven development method. In order to solve this problem, we apply component-based software technologies that have been mainly developed on software implementing level to modeling level. In the proposed method, functionally relevant model elements are packed as a model component, and modeling software is carried out by associating it with each model component. The role of model components becomes clear by introducing the concept of a component, and the reuse of model components rises by externalizing the dependency between components. In addition, flexible model transformation rules linked to the role of model components can be designed. As a result, it is possible to automatically generate more source code. The validity of the proposed method has been proved by application experiments.**

Keywords-model; component-based development; object oriented design; UML; MDA

I. INTRODUCTION

Model driven development (MDD) method which is based on model driven architecture (MDA) [1] draws attention as a technique that can flexibly deal with changes of business logics or implementing technologies in the field of system development. Its core data are models that serve as design diagrams of software. It includes a transformation to various kinds of models and an automatic source code generation from the models. These models themselves are expected to be reused [2]. By reusing models, you can also reuse the knowledge that does not depend on platforms. It can improve software development efficiency. For that purpose, it is necessary to pack the highly relevant model elements into a function and to clarify a reuse unit [3].

On the programming level, there is a development technique called component-based software method [4] that especially aims at reusing the source code. A component is a set of highly relevant and reusable program parts. The component-based method develops software by combining components. It has been mainly improved in the programming field, and various development frameworks are now put into practice. In recent years, component-based modeling [5] which applies the concept of components developed on programming to software models has been advocated. The software models treated by this component-based modeling and the models treated by MDD are the same.

You can develop the component-based modeling further and apply it to MDD. According to this idea, this paper proposes a method that makes model elements loose coupling by introducing a component of the component-based technologies to MDD. It also describes a technique to design automatic generation rules that is one of the features of MDD. Finally, this study aims at increasing the efficiency of MDD by the proposed software development method.

The structure of this paper is shown below: Section II describes component-based software technologies, and Section III explains the proposed method of this research. In Section IV, we show the results of application experiments in order to confirm the validity of the proposed method. Finally, Section V describes conclusions and future subjects.

II. COMPONENT-BASED SOFTWARE TECHNOLOGIES

Fundamentally, a component is not used alone, but it is used in order to build software by combining it with other components or programs. This study uses software patterns called Inversion of Control (IoC) and Dependency Injection (DI).

A. Inversion of Control

IoC is a kind of software architecture, which frameworks or containers actively call components [6]. On the contrary, components do not call other cooperative components, but they only have references to their interface instead. Components do not need to know other cooperative components at the time of software implementation. This promotes interface programming and stimulates loose couplings of software.

B. Dependency Injection

By realizing IoC through interfaces, you need a mechanism to get the required components at the time of execution. DI [7] is a technique to realize the mechanism. Dependency is injected outside components. The dependency is injected outside the components and described apart from the source code expressing business logics. A container managing the life cycle of components analyzes the dependency at the time of execution and specifies the components to cooperate with according to the dependency. It is not necessary to describe the process of acquiring external components in the source code which realizes business logics. Therefore, IoC is realized.

III. Proposed Method

Dependency between components is externalized by introducing IoC into a model level, and loose coupling between components can be also promoted. The proposed method adopts this approach in MDD. It aims at improving the efficiency of MDD by increasing the efficiency of modeling, reusing models, and defining an externalized dependency between components. This section explains the modeling process suggested in the proposed method by using Unified Modeling Language (UML) diagrams. The position of the proposed method is shown in Fig. 1. Fig. 2 shows a flow of development cycle in the proposed method.

A. Modeling

Modeling is an approach for introducing IoC in class relations, and messages passing through components and container interfaces. This approach is close to the Catalysis approach [3], which decides on component interfaces in advance and models the inside of the components independently. However, the proposed method is not necessary to know external components because it adopts the IoC, and it differs from the Catalysis approach in this point. The proposed method condenses software modules that are functionally connected as a model component. It is required to decide component granularity by introducing the concept of components into upstream processes from the implementing level. Components are classified by the granularity of practical software examples as follows:

1) Business Component: It corresponds to one of business processes like an order receipt, estimate. It is realized by compounding business functional components.

2) Business Functional Component: It is a business element unit like accepting order receipt, replying estimation request. A service in Service Oriented Architecture (SOA) is located here.

3) System Functional Component: It is a system part like registration, updating and reference of information. A business functional component is created using the system functional components.

The proposed method targets at the system functional components among various granularities.

B. Externalizing Dependency of Model Elements

The proposed method also designs a model for defining dependency between model components. This is referred to as an external cooperation model. A component diagram of UML is used for modeling of external cooperation models. The component diagram, which shows a static structure of software, can be divided into demand interfaces and supply interfaces. It can describe an interface and the dependency, including its directivity [8]. The proposed method calls an internal structure model that shows the inside of the model component structure. To design the internal structure model, we use a class diagram of UML. However, the class diagram makes use of internal classes, interfaces, libraries, and demand/supply interfaces that belong to the component. It does not directly use external elements to keep IoC. The external cooperative models are defined outside of the

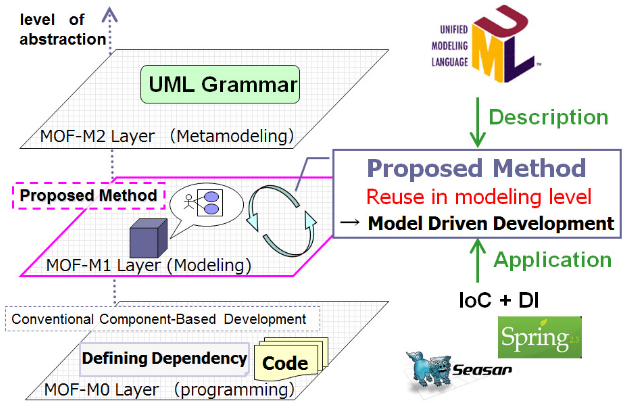


Figure 1. Position of the proposed method.

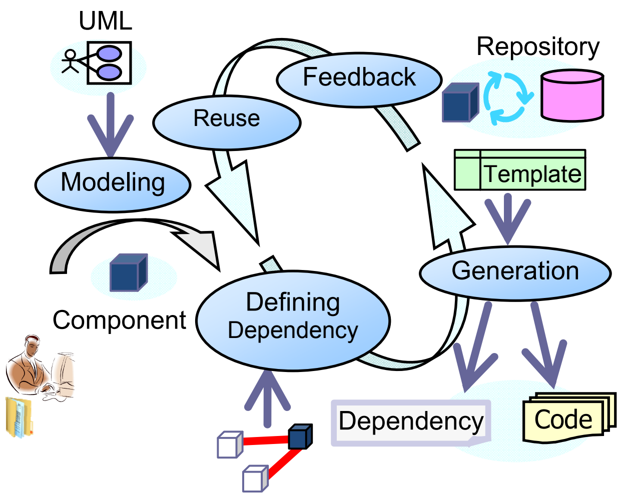


Figure 2. Cycle of software development.

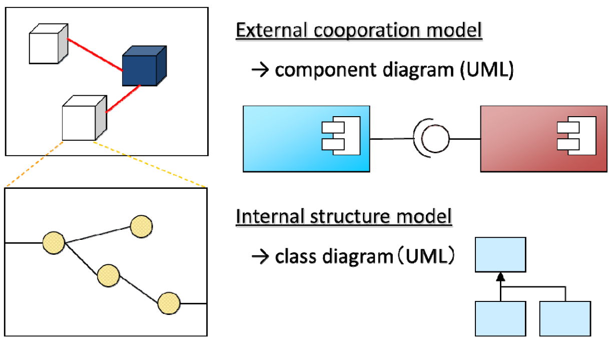


Figure 3. Dependency of model elements.

components. The concept of IoC is introduced into a model level, and externalizing dependency between components is achieved. Fig. 3 shows these concepts.

C. *Selective Design of Model Transformation Rules*

In order to transform models to texts like source code, you need to design a template describing transformation rules from the model to the texts. The proposed method makes the role of components clear by introducing the concept of a component and it enables to design a template dedicated to a component. For example, prominent software patterns like analysis patterns [2] and design patterns [9] [10] [11] can be used and employed as model components or transformation rules. The proposed method does not limit the type of the system for development. It is advantageous that a developer can give role information freely to model components or transformation rules.

It is very difficult to design a general-purpose template in a platform. If a template is designed for a specific project [12], it will become hard to divert it to other projects. The proposed method designs a template for each component because it is separated according to a function of the component. For this reason, the template design becomes more flexible than ever. It makes use of the externalizing dependency and no need to write special code for DI in source code.

IV. Application Experiments

The proposed method is applied to a sample system to confirm the effectiveness of the proposed method.

A. *Experimental Overview*

The sample system is a typical three layer client server type of Web application. The main roles, implementing methods, and applied design patterns are shown in the following:

1) Business Logic Layer: It corresponds to functional requirements of the Web application. It receives inputs from a controller and sends processing results to a presentation layer. In addition to service components, a naming service by Java Naming and Directory Interface (JNDI) is realized in order to make possible Remote Method Invocation (RMI) of procedure. Service Locator, Business Delegate, Singleton, and Proxy are used as design patterns.

2) Data Access Layer: It abstracts a persistent data system, such as a database or a file system. It is a uniform window to deal with the data system. Data Access Object, Factory Method, Abstract Factory, and Facade are used as design patterns.

3) Entity: It is a class group expressing the data that serve as persistent objects among the object classes in a problem domain. Data Transfer Object (Value Object), Composite Value Object, and Value Object Assembler are used. In addition, platform specifications are shown in the following:

- Programming Language: Java Development Kit 5.0.
- Application Server: Glassfish Open Source Edition v3.
- Database Management System: MySQL 5.1

Many MDA tools generate skeleton code from class diagrams of UML. For comparison, these tools are regarded as conventional methods. A standard sample of Acceleo [13] is used in this experiment. The Acceleo sample has an automatic generation function of Java source code from class diagrams designed by UML modeling tool UML2 of Eclipse. We compare three kinds of experimental data: the source code automatically generated by the proposal method, the source code generated by the conventional method, and the final source code. The last one is composed of the source code resulting from the proposed method and hand coded additional source code. These experimental data are respectively called "proposed method", "conventional method", and "finished goods".

B. *Template Design*

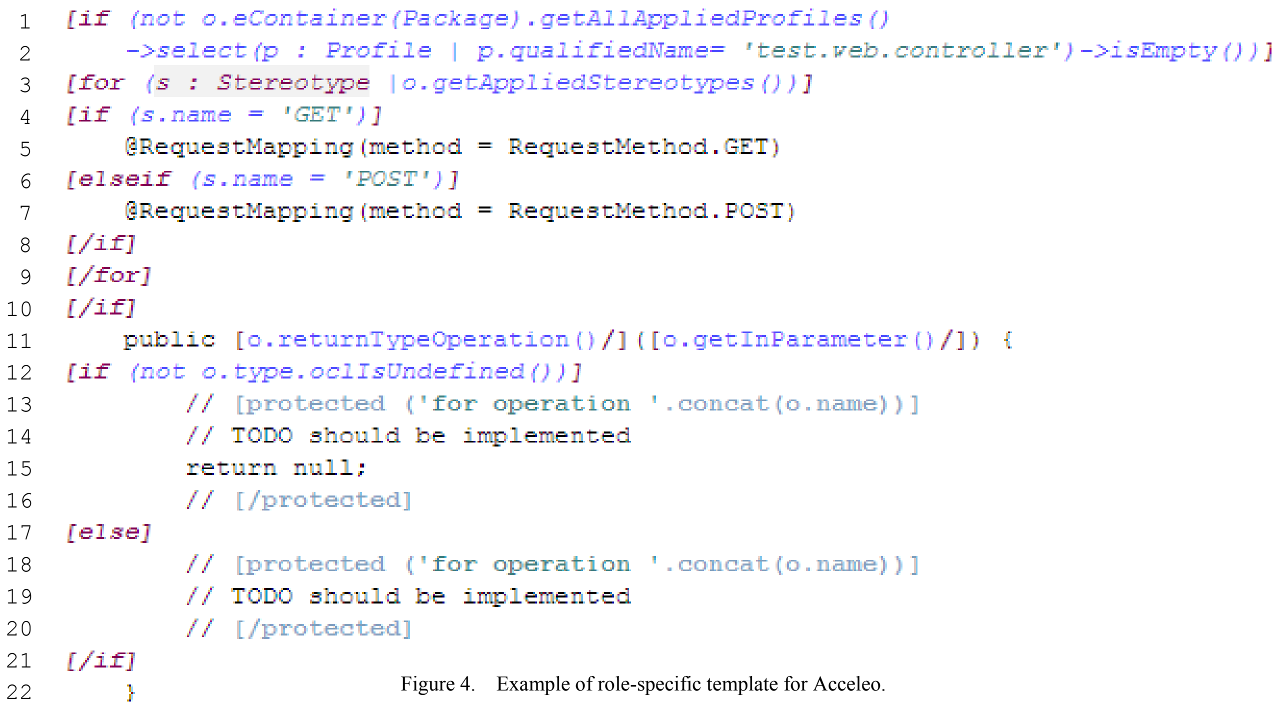
Fig. 4 is an example of the dedicated template that generates the code of a class method, implemented by Java+Spring Framework, for Web page controller models. This model component has a Web page controller profile, as confirmed by the lines 1-2. An appropriate function is given to a class method by detecting the applied stereotype as shown in the lines 3-9. This template is stored in the model component of the Web page controller. The proposed method enables to design a specialized template in a UML profile. The template is also united with a role, but it is not based on a field. In addition, the automatic generation rate of code also becomes higher.

The templates are prepared by roles, such as a template corresponding to design patterns, controllers in a client server type, and database access. A template reads information on a model and is a medium that generates source code. The automatic generation is carried out by each model component. The source code is merged with information on the internal structure model, the external cooperation model, and the UML profile.

C. *Evaluation by Abstract Syntax Tree*

This experiment measures the number of Abstract Syntax Tree (AST) nodes in order to evaluate the quantity of the source code generated automatically according to procedure of the MDD. The AST syntactically analyzes the source code and expresses it in a directed tree. There are two advantages of investigating AST nodes: The first one is that it may be easier to reflect the actual processing of a program than with conventional indexes, such as Line of Code (LOC): number of source code lines. The second one is that it can investigate a type of nodes that constitutes the AST. This information is useful when knowing the structure of a program. On the contrary, LOC is a simple numerical value and cannot know the structure of the program.

This experiment uses Eclipse Java Development Tools (JDT) as an API that builds AST from Java source code. In addition, language specification is AST-Java Language Specification3 (JLS3). JLS3 is equivalent to Java Development Kit 5.0. When LOC is calculated, you may



```
[if (not o.eContainer(Package).getAllAppliedProfiles()
    ->select(p : Profile | p.qualifiedName= 'test.web.controller')->isEmpty())]
[for (s : Stereotype |o.getAppliedStereotypes())]
[if (s.name = 'GET')]
    @RequestMapping(method = RequestMethod.GET)
[elseif (s.name = 'POST')]
    @RequestMapping(method = RequestMethod.POST)
[/if]
[/for]
[/if]
    public [o.returnTypeOperation()/]([o.getInParameter()/]) {
[if (not o.type.oclIsUndefined())]
        // [protected ('for operation '.concat(o.name))]
        // TODO should be implemented
        return null;
        // [/protected]
[else]
        // [protected ('for operation '.concat(o.name))]
        // TODO should be implemented
        // [/protected]
[/if]
    }
```

Figure 4. Example of role-specific template for Acceleo.

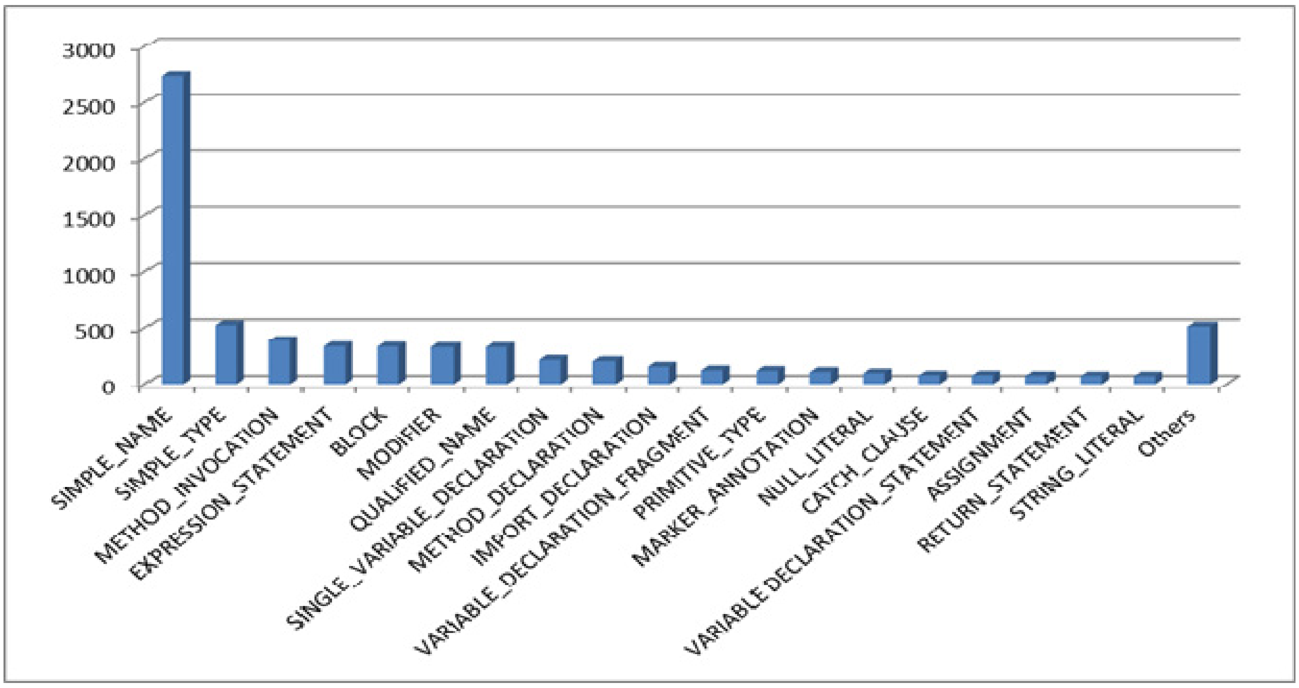


Figure 5. Breakdown of AST nodes generated by the proposed method.

disregard lines that are unrelated to the program execution such as comment lines. The value measured on the condition is called a logical LOC. Like this, this experiment eliminates LINE COMMENT, BLOCK COMMENT, and JAVADOC that express comments when you add up AST nodes.

Fig 5 indicates each AST node's category of the proposed method and the number of nodes belonging to the category. All graphical models that serve as inputs of the automatic generation tools are the same. The total numbers of AST nodes for the finished goods, the proposed method, and the conventional method are 7307, 6859, 2200, respectively. The execution time of the conventional method is really shorter than the one of the proposed method. However it could generate the third of AST nodes of the proposed method, and includes many errors in the generated source code.

D. Discussion

The number of AST nodes is remarkably increased by the proposed method in comparison with the conventional method. It turns out that class methods occupy a large proportion in the source code of a system. The proposed method is effective in MDD from this point. In addition, generated AST nodes of the proposed method are investigated in detail as compared with the conventional method. SIMPLE NAME nodes are ranked number one. They express class names, field names, class method names, variable names, and so on, as it appears everywhere in the source code. They are located in terminal nodes in AST. Therefore, there are more SIMPLE NAME nodes than other nodes in both the proposed method and the conventional method. From this point of view, it cannot show advantages of the proposed method. The other AST nodes that appear in class methods are investigated except the SIMPLE NAME nodes. The total of the increasing amount of these AST nodes, except for SIMPLE NAME nodes, is 1508 and it occupies about 32.4% of the whole increasing amount 4659. It becomes about 56.7% if the SIMPLE NAME nodes are eliminated. Fig. 6 is a bar graph in which incremental values of automatic generated code of the proposed method are ranked in descending order in comparison with the conventional method by the category of AST nodes. However, this graph omits the SIMPLE NAME nodes and the nodes which incremental amount are zero.

Regarding applied design patterns, it is easy to design templates for Service Locator, Business Delegate, and Data AccessObject. These patterns are more commonly used to describe the bodies of class methods. The proposed method is superior in dealing with design patterns. The readability of the generated source code is expected to be high. The readability is one of the elements that is not described in models but appears in source code. It is easy to describe it as

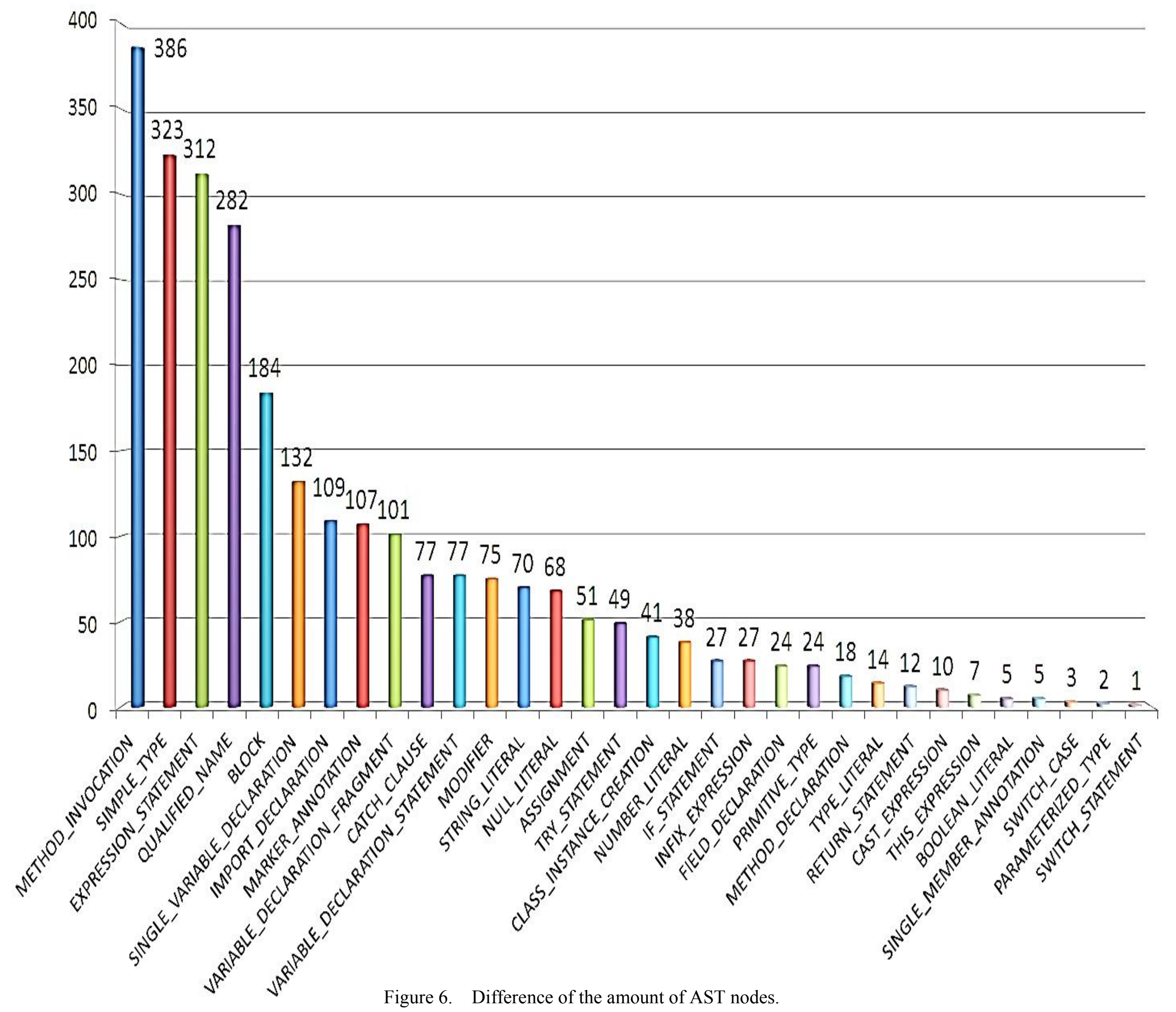


Figure 6. Difference of the amount of AST nodes.

coding conventions in templates. Although it is necessary to design templates for every role of the model components by the proposed method, they are independent from fields and they are reusable for other projects.

AndroMDA [14], Software Factories [15], and UML Components [16] do not give class methods. They are almost the same as with the conventional method. BridgePoint [17] that gives class methods by an action description language makes a little difference with the proposed method about automatic generation rates of source code. But it does not develop certain patterns using the concept of IoC and DI. It is hard to design templates and to build them individually. Some corrections are needed to reuse the component models and templates for other examples, and it has a low chance of reuse either.

V. Conclusions

This paper has proposed a software development method that applies component-based technologies to MDD. The proposed method was compared with the conventional method using a sample Web system. This method is able to automatically generate more than three times the amount of AST nodes compared with the conventional method. The proposed method can describe information concerning class methods in the templates. Especially, describing functions of source code or main parts of methods in templates improves the rate of automatic code generation. This fact shows the advantages of the proposed method concerning the generation efficiency of class methods, by designing templates linked to the role of model components.

On the other hand, insufficient parts of the proposed method as compared with the finished goods are mainly business logics. They are class methods that belong to business components. It is difficult to build templates for these parts because they are easily affected by requirement specifications. As a result, the automatic generation rate is low. As a future subject, the proposed method should deal with a different granularity, like business components. In addition, a further extension could be to take into account dynamic behavior diagrams.

Acknowledgment

This work was supported in part by JSPS KAKENHI Grant Number 24560501.

References

[1] S.J. Mellor, K. Scott, A. Uhl, and D. Wiese, MDA Distilled, Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, 2004.

[2] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley, 1997.

[3] D.F. D'souza and A.C. Wills, Objects, Components, and Frameworks with UML: The Catalysis Approach, lavoisier.fr, 1998.

[4] I. Crnkovic, “Component-based software engineering - new challenges in software development,” Software Focus, Vol.2, Dec. 2001, pp. 127-133, doi:10.1002/swf.45.

[5] G. Gossler and J. Sifakis, “Composition for component-based modeling,” Science of Computer Programming, Vol.55, Mar. 2005, pp. 161-183.

[6] D. Rosenburg and M. Stephens, Use Case Driven Object Modeling with UML: Theory and Practice, Apress, New York, 2007.

[7] Spring framework, http://www.springsource.org/, [retrieved: April, 2014].

[8] Object Management Group, OMG unified modeling language superstructure specification, V2.1.2, 2007.

[9] E. Gamma, R. Helm, R. Jhonson, and J. Vlissides, Design Patterns: Eelements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[10] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: A System of Patterns. Wiley, 1996.

[11] D. Alur, D. Malks, and J. Crupi, Core J2EE Patterns: Best Practices and Design Strategies, Prentice Hall, 2001.

[12] K. Matsumoto, T. Maruo, M. Murakami, and N. Mori, “A graphical development method for multiagent simulators,” in Modeling, Simulation and Optimization: Focus on Applications, S. Cakaj, Ed. Vukovar: In-Tech, 2010, pp. 147-157.

[13] Acceleo, http://www.eclipse.org/acceleo/, [retrieved: April, 2014].

[14] AndroMDA, http://www.andromda.org/, [retrieved: April, 2014].

[15] J. Greenfield and K. Short, “Software factories: assembling applications with patterns, models, frameworks and tools,” Proc. Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Oct. 2003 pp. 16-27.

[16] J Cheesman and J Daniels, UML Components. Addison-Wesley, 2001.

[17] BridgePoint, http://www.bridgepoint.eu/, [retrieved: April, 2014].