Intelligent Software Development Method Based on Model Driven Architecture

Keinosuke Matsumoto, Kimiaki Nakoshi, and Naoki Mori
Department of Computer Science and Intelligent Systems
Graduate School of Engineering, Osaka Prefecture University
Sakai, Osaka, Japan
email: {matsu, nakoshi, mori}@cs.osakafu-u.ac.jp

***Abstract*—Recently, Model Driven Architecture (MDA) has attracted attention in the field of software development. MDA is a software engineering approach that uses models to create products such as source code. On the other hand, executable Unified Modeling Language (UML) consists of activities, common behavior, and execution models; however, it has not been effectively transformed into source code. This paper proposes a method for transforming executable UML and class diagrams with their associations into source code. Executable UML can describe a system's behavior well enough to be executed; however, it is very difficult for executable UML to handle system data. Therefore, the proposed method uses class diagrams for this purpose. The method can create models independent of platforms, such as programming languages. The proposed method is applied to a system, where Java and C# code was generated from system models, which were generated using an executable UML model; in addition, development costs are evaluated. As a result, it is confirmed that this method can significantly reduce costs when models are reused.**

***Keywords*—executable UML; activity diagram; model driven architecture; UML.**

I. INTRODUCTION

This paper is based on the study presented at INTELLI 2017 [1]. In today's software development environment, software reuse, modification, and migration of existing systems have increased at a greater pace than new development. According to an investigative report [2] by the Information-Technology Promotion Agency (IPA), reuse, modification, and migration of existing systems account for approximately 73.3% of software development and new development accounts for 26.2% as shown in Fig. 1. Many software bugs enter the upper processes, such as requirement specification, system design, and software design. However, bugs are mostly discovered in lower processes, such as the testing process. The need to detect bugs upstream is gaining priority. Under such a situation, software developers require a development technique that is easy to reuse and that adjusts to changes in implementation technique. Model driven architecture (MDA) [3] is attracting attention as an approach that generates source code automatically from models that are not influenced by implementation [4–6]. Its core data are models that serve as design diagrams of software. It includes a transformation to various types of models and automatic source code generation based on the models. Therefore, it can directly link software design and implementation.

The final goal of MDA is to generate automatically executable source code for multiple platforms. For that purpose, it is necessary to make the architecture and behavior of a system independent from platforms, e.g., a Platform Independent Model (PIM) that does not depend on platforms, such as programming languages. Executable Unified Modeling Language (UML) [7][8] is advocated as this type of model as it expresses all actions for every type of processing, and expresses input and output data by a pin in activity diagram, which is one of various UML [9] diagrams. The source code for various platforms can be generated from one MDA-type model because processing and data can be transformed for every platform if executable UML is used.

In this study, a method is proposed that generates source code automatically from executable UML. It is very difficult for executable UML to handle a system's data. To solve this problem, this paper proposes a modeling tool that associates an executable UML with class diagrams and acquires data from them. It can treat not only data, but can introduce the hierarchical structure of class diagrams in executable UML. If the platform of future systems, such a programing language, is changed, software developers cannot reuse existing source code, but they can reuse UML models to automatically generate source code in the new programming language.

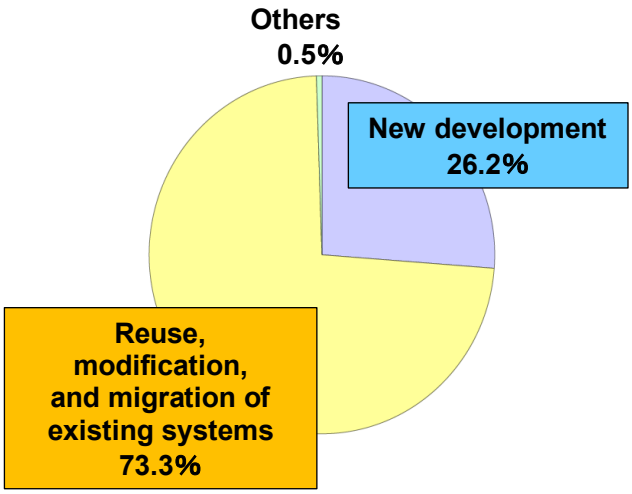

Figure 1. Percentage of software development projects.

The contents of this paper are as follows. In Section II, background of this study is described. In Section III, the proposed method is explained. In Section IV, the results of application experiments confirm the validity of the proposed method. Finally, in Section V, the conclusions and future work are presented.

II. BACKGROUND

This section describes background of this study that makes use of Acceleo and executable UML.

MDA's core data are models that serve as software design drawings. The models are divided into platform dependent and independent models. Auto source code generation tools (for example, Acceleo [10]) transform models from PIM to a Platform Specific Model (PSM) that depends on a platform, and generates source code automatically. Transformation of the PIM is important and can generate source code in various platforms by applying different transformation rules to each platform. Acceleo is a plug-in of the integrated development environment Eclipse [11], and a code generator that translates MetaObject Facility (MOF) [12] type models into source code on the basis of code transformation rules called a template. Almost all source code generation tools like Acceleo can translate the models directly, but the tools have many constraints. For example, they can only generate skeleton code. In addition, they cannot hold and calculate data. Therefore, they cannot recognize what types of model elements have been read. It is impossible to search the connections between nodes by using graph theory. When branches and loops of activity diagrams are transformed, the generation tools have a problem in that they cannot appropriately transform them because they do not understand the environment.

Executable UML is a model based on activity diagrams, as shown in Fig. 2. It has the following features:

- An action is properly used for every type.
- Input and output data of each action are processed as a pin, and they are clearly separated from the action.
- A model library that describes the fundamental operations in a model is prepared.

Each type of action has respectively proper semantics, and transformation with respect to each action becomes possible by following the semantics. The type and semantics of an action used in executable UML are as follows.

1) ValueSpecificationAction: Outputs a value of primitive type data such as an integer, real number, character string, or logical value.

2) ReadStructuralFeatureAction: Reads certain structural characteristics. For example, it is used when the property of class diagrams is read.

3) ReadSelfAction: Reads itself.

4) CallOperationAction: Calls methods in class diagrams.

5) CallBehaviorAction: Calls behaviors in behavior diagrams.

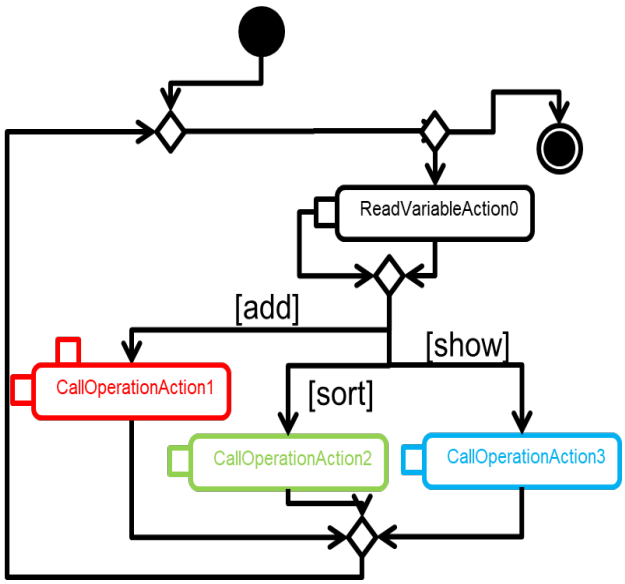

Figure 2. Example of executable UML.

6) AddVariableValueAction: Adds a value to the variable or replaces the variable with its value.

7) ReadVariableAction: Reads a variable or generates one.

8) CreatObjectAction: Creates a new object.

The model library consists of the Foundational Model Library, Collection Classes, and Collection Functions. The contents of the model library are shown below.

1) Foundational Model Library: Offers primitive type data, and their behaviors (four arithmetic operations, comparison, etc.) and all input-output relationships.

2) Collection Classes: Offers the collection classes of Set, Ordered Set, Bag, List, Queue, Dequeue, and Map.

3) Collection Functions: Offers the methods (add, delete, etc.) of the collection class.

The model library is used by calling CallOperationAction or CallBehaviorAction.

III. PROPOSED METHOD

This section explains the technique of transforming executable UML to source code. Although executable UML is useful, this model has not been used effectively for automatic generation of source code. Moreover, the handling of data is inadequate if using only executable UML. To solve this problem, a method is proposed for generating source code automatically from executable UML. The method utilizes a modeling tool that associates executable UML with class diagrams. If executable UML requires data, the method retrieves the data from associated class diagrams.

The outline of the proposed method is shown in Fig. 3. Skeleton code is transformed from class diagrams by using Acceleo templates [13] for classes. The skeleton code consists only of class names, fields, and methods that do not have specific values of data. Data and a method to

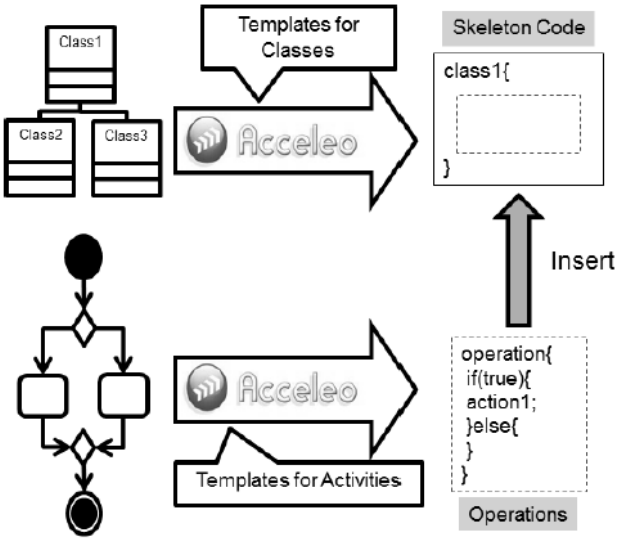

Figure 3. Schematic diagram of the proposed method.

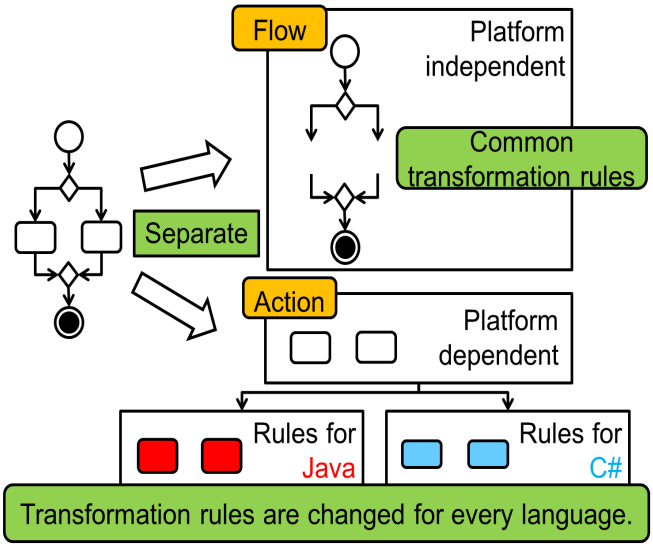

Figure 4. Separation of executable UML.

manipulate the data are automatically generated from executable UML afterward. Since data is associated with class diagrams, other methods and data in the classes are acquirable using this association. Papyrus UML [14] was used for the associations among these models.

A. *Transformation from Class Diagrams to Skeleton Code*

UML to Java Generator [15] was used to convert the transformation rules from class diagrams to skeleton code. These rules generate the following.

- Connection of inheritance or interface
- Field variables and methods (e.g., getter and setter)
- Names and parameters of member functions

This is a template for Java. When transforming models to C#, several additional changes are required, such as deletion of constructors and addition of ":" for the inheritance relationship.

B. *Transformation from Executable UML to Source Code*

Executable UML is based on activity diagrams, which consist of actions, data, and their flows. Although transform rules of actions and data differ from platform to platform, the flows are fundamentally the same. Therefore, transformation of flows is separated from transformation of actions and data as shown in Fig. 4. Flows decide the order of transformation of actions and data. This separation can flexibly transform one model to the source code of multiple platforms. The transformation flow of executable UML is described as follows.

1) *Transformation of flows*

A flow of executable UML is shown by connecting nodes, which include actions and data, with an edge. However, neither a branch nor loop is transformed by only connecting nodes along the flow. On transforming a decision or merge node used for a branch or loop, the proposed

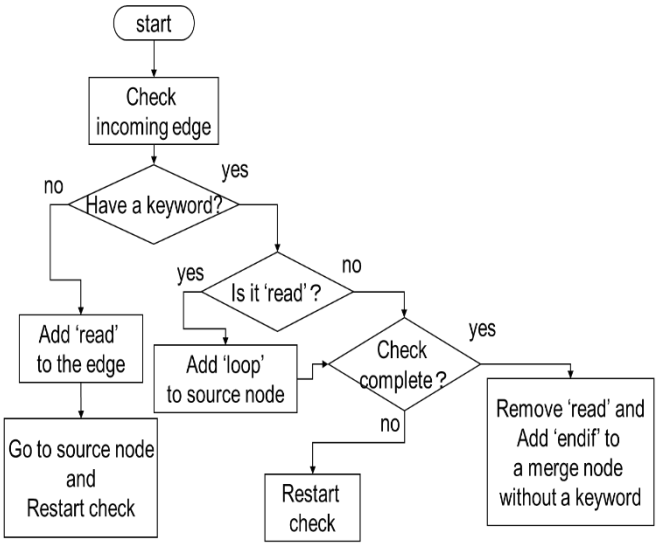

Figure 5. Search algorithm.

method searches a part of the executable UML near the node and provides an appropriate keyword to a connecting node and edge. The method transforms them according to keywords. The keywords given to model elements are shown below.

a) finish: Indicates a node or edge whose processing has finished.

b) loop: A decision node in the entrance or exit of a loop.

c) endif: A merge node at the end of a branch.

d) read: An edge is being searched.

The flow of the search is shown in Fig. 5 and its algorithm is as follows.

(1) Follow an edge that is not searched in the reverse direction of its arrow.
(2) If an edge and node have not been searched, issue the keyword 'read'.
(3) If a node has the keyword 'read', replace the keyword with 'loop'. If the node is a decision node, assign the keyword 'loop' to a searched edge going out from the decision node.
(4) Repeat the processing of (1) –(3) until there is no edge remaining to be searched.
(5) Remove 'read' after the last edge. If there is a merge node that does not have keyword 'loop', 'endif' will be assigned to it.

2) Transformation of actions and data

Transformation rules of actions and data are prepared for every platform. In executable UML, an action is properly used for every type of processing, and a transformation rule may be defined per action. The flow of transformation processing is as follows.

(1) If a node is an action, it will be transformed and assigned the keyword 'finish'. Processing will move to the next node. If the action has an input pin, its flow will return and the objects and actions at the starting point of this flow will be transformed.
(2) If a node is a decision node and it has the keyword 'loop', it will be transformed by rules for a loop. If the decision node has no keyword, it will be transformed by the rules for a branch. In addition, the nodes and conditional expressions are retrieved from the connecting edges. When 'finish' keywords are assigned to these nodes and edges, processing will move on to the next nodes.
(3) If a keyword is assigned to a merge node, it will be transformed according to the rules of the keyword.

Corresponding relationships of actions between Java and C# are shown in Table I. The upper row of ReadStructural FeatureAction in Table I is the case where <*variable*> is specified, and the lower row is the case where it is not specified.

In addition, corresponding relationships of model libraries between Java and C# are shown in Table II. ReadLine and WriteLine are model libraries for input and output. List.size, List.get, and List.add are prepared by Collection Functions, and they are used for output of list capacity, extraction of list elements, and addition of list elements, respectively. Primitive Functions are operations prepared in the primitive type. Collection Functions are used by calling CallOperationAction, and all other functions are used by calling CallBehaviorAction. Variables inputted by pins and operators defined in the library are shown in italics surrounded by < >.

IV. APPLICATION EXPERIMENTS

As an experiment that verifies operation and the development costs of created templates, the system shown below (Figures 6 and 7) was developed using the proposed method. The system receives three commands for adding data to a list, sorting list elements, and outputting data. This system was described by executable UML and class

TABLE I. ACTIONS AND THEIR APPLICATIONS TO EACH LANGUAGE.

Action	Java	C#
CreateObjectAction	new <*object*>	new <*object*>
ReadSelfAction	this	this
ValueSpecification Action	<*value*>	<*value*>
ReadStructural FeatureAction	<*object*>.<*variable*>	<*object*>.<*variable*>
	(<*resultType*>) <*object*>	<*resultType*>.Parse (<*object*>)
CallOperationAction	<*target*>.<*operation*> (<*parameter*>)	<*target*>.<*operation*> (<*parameter*>)
AddVariableValue Action	<*variable*>=<*value*>	<*variable*>=<*value*>

TABLE II. MODEL LIBRARY ELEMENTS AND THEIR APPLICATIONS.

Model Library	Java	C#
ReadLine	(new BufferedReader(new InputStreamReader(System. in))).readLine()	Console.ReadLine()
WriteLine	System.out.println(*value*)	Console.WriteLine(*value*)
List.size	<*target*>.size()	<*target*>.Count
List.get	<*target*>.get(<*index*>)	<*target*>[<*index*>]
List.add	<*target*>.add(<*data*>)	<*target*>.Add(<*data*>)
Primitive Functions	<*x*><*function*><*y*>	<*x*><*function*><*y*>

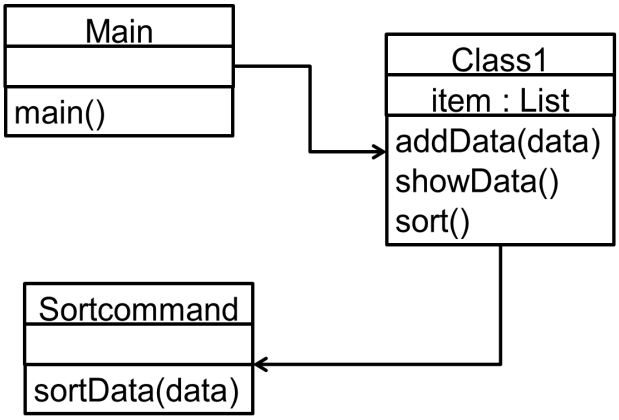

Figure 6. Class diagram of example system.

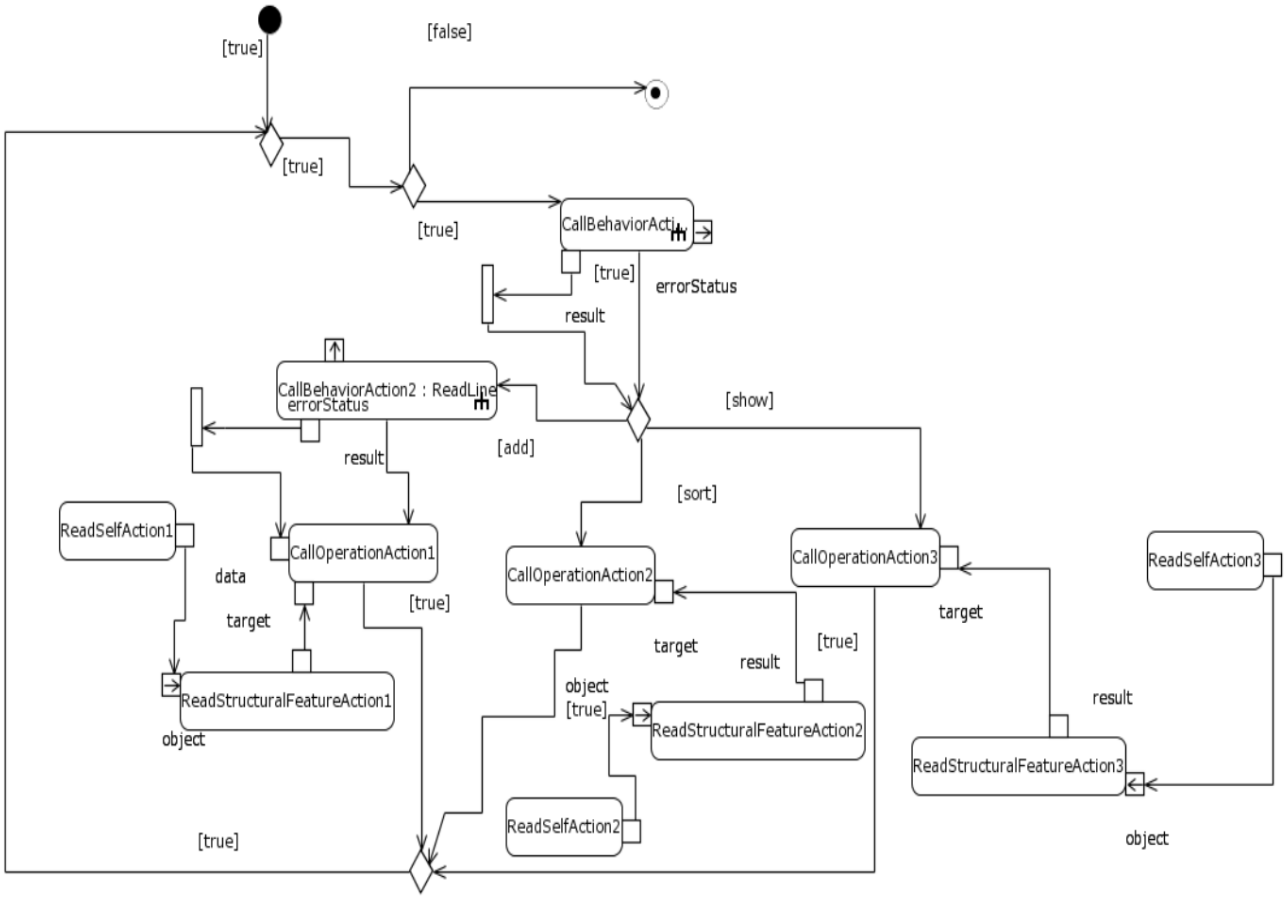

Figure 7. Total system behavior of example system.

```java
package Package1;
import java.util.ArrayList;
public class Class1 {
    private Sortcommand sortcommand=new Sortcommand();
    private List item=new List();
    public Class1() {
        super();
    }
    public void addData(String data) {
        this.item.add(data);
    }
    public void showData() {
        System.out.println(this.item);
    }
    public void sort() {
        this.sortcommand.sortData(this.item);
    }
    public Sortcommand getSortcommand() {
        return this.sortcommand;
    }
    public void setSortcommand(Sortcommand newSortcommand) {
        this.sortcommand = newSortcommand;
    }
    public List getItem() {
        return this.item;
    }
    public void setItem(List newItem) {
        this.item = newItem;
    }
}
```

Figure 8. Generated Java code of Class1.

```csharp
using System;
using System.Collections.Generic;
namespace Package1
{
    public class Class1 {
        private Sortcommand sortcommand=new Sortcommand();
        private List item=new List();
        public  void sort() {
            this.sortcommand.sortData(this.item);
        }
        public  void showData() {
            Console.WriteLine(this.item);
        }
        public  void addData(String data) {
            this.item.Add(data);
        }
        public Sortcommand getSortcommand() {
            return this.sortcommand;
        }
        public void setSortcommand(Sortcommand newSortcommand) {
            this.sortcommand = newSortcommand;
        }
        public List getItem() {
            return this.item;
        }
        public void setItem(List newItem) {
            this.item = newItem;
        }
    }
}
```

Figure 9. Generated C# code of Class1.

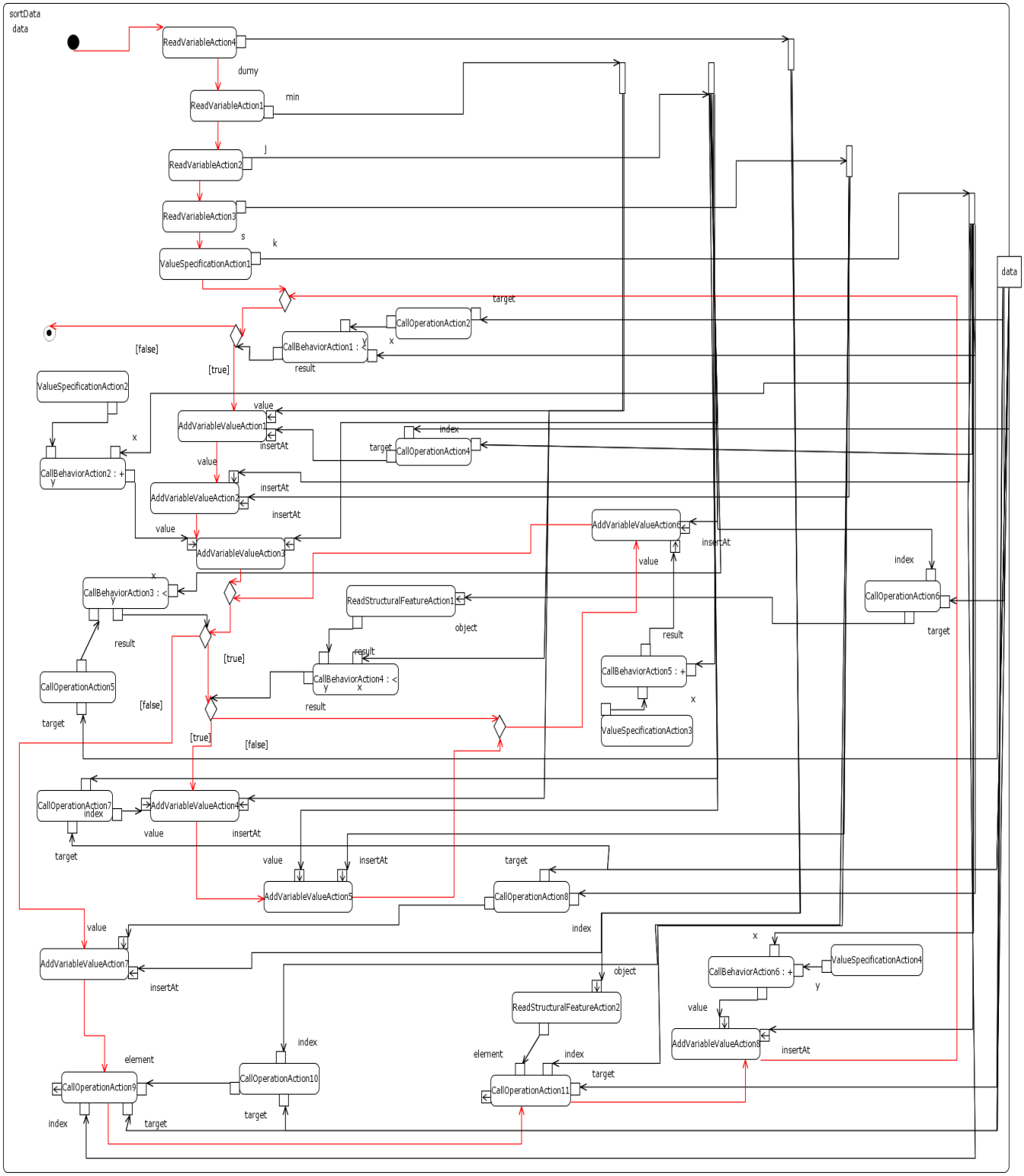

Figure 10. Executable UML of sortData.

```
[template public trs4action(act : Action)]
[if (act.output->notEmpty())]
[let nexN : Sequence(ActivityNode) = act.output.outgoing.target]
[if (nexN->any(oclIsTypeOf(ForkNode))->notEmpty())]
[act.output.type.genType()+' '/][nexN->any(oclIsTypeOf(ForkNode)).name/]=[/if][/let][/if][if (act.oclIsTypeOf(CallBehaviorAction))]
[callBeh(act.oclAsType(CallBehaviorAction))/][elseif (act.oclIsTypeOf(ValueSpecificationAction))]
[valuespe(act.oclAsType(ValueSpecificationAction))/][elseif (act.oclIsKindOf(ReadVariableAction))]
[act.oclAsType(ReadVariableAction).genReadvar()/][elseif (act.oclIsKindOf(CallOperationAction))]
[act.oclAsType(CallOperationAction).callOpe()/][elseif (act.oclIsKindOf(AddVariableValueAction))]
[act.oclAsType(AddVariableValueAction).genAddvar()/][elseif (act.oclIsKindOf(ReadStructuralFeatureAction))]
[act.oclAsType(ReadStructuralFeatureAction).genReadstr()/][elseif(act.oclIsTypeOf(ReadSelfAction))]
[act.oclAsType(ReadSelfAction).genReadSelf()/][else]
[act.name/]([for (it : String | act.input.invalue().genInput())separator (',')][it/][/for])[/if]
[/template]
[template  callOpe(ope : CallOperationAction)]
[ope.target.invalue().genInput()/].[ope.operation.genOpe()/]([for (it : String | ope.argument.invalue().genInput())separator (',')][it/][/for])
[/template]
[template public genOpe(ope : Operation)]
[if (ope.class.ancestors(Package).name->includes('Alf'))]
[if (ope.class.ancestors(Package)->reject(oclIsTypeOf(Model)).name->includes('CollectionClasses'))]
[if (ope.name.matches('at'))]
get[elseif(ope.name.matches('replaceAt'))]
set[else]
[ope.name/][/if][else]
[ope.name/][/if][else]
[ope.name/][/if]
[/template]

[template public callBeh(act : CallBehaviorAction)]
[if (act.behavior.ancestors(Package).name->includes('FoundationalModelLibrary'))]
[if (act.behavior.ancestors(Package)->reject(oclIsTypeOf(Model)).name->includes('BasicInputOutput'))]
[if (act.behavior.name='WriteLine')]
System.out.println([act.argument.invalue().genInput()/])
[elseif (act.behavior.name='ReadLine')]
[let pin  : Pin = act.result->any(oclIsKindOf(Pin))]
(new BufferedReader(new InputStreamReader(System.in))).readLine()[/let][/if][elseif (act.behavior.ancestors(Package)->reject(oclIsTypeOf(Model)).name
[if (act.argument->size()=2)]
[act.argument->any(name='x').invalue().genInput()/][act.behavior.name/][act.argument->any(name='y').invalue().genInput()/][else]
[act.behavior.name/]([for (it : String | act.argument.invalue().genInput()) separator (',')][it/][/for][/if][/if][/if]
[/template]
[template public valuespe(act : ValueSpecificationAction)]
[if(act.value.oclIsTypeOf(LiteralString))]
"[act.value.stringValue()/]"[elseif (act.value.oclIsTypeOf(LiteralInteger))]
[act.value.integerValue()/][elseif (act.value.oclIsTypeOf(LiteralBoolean))]
[act.value.booleanValue()/][elseif (act.value.oclIsTypeOf(LiteralReal))]
[act.value.realValue()/][/if]
[/template]
[template public genReadSelf(act : ReadSelfAction)]
this
[/template]
```

Figure 11. A portion of the template for Java.

```
[template public trs4action(act : Action)]
[if (act.output->notEmpty())]
[let nexN : Sequence(ActivityNode) = act.output.outgoing.target]
[if (nexN->any(oclIsTypeOf(ForkNode))->notEmpty())]
[act.output.type.genType()+' '/][nexN->any(oclIsTypeOf(ForkNode)).name/]=[/if][/let][/if][if (act.oclIsTypeOf(CallBehaviorAction))]
[callBeh(act.oclAsType(CallBehaviorAction))/][elseif (act.oclIsTypeOf(ValueSpecificationAction))]
[valuespe(act.oclAsType(ValueSpecificationAction))/][elseif (act.oclIsKindOf(ReadVariableAction))]
[act.oclAsType(ReadVariableAction).genReadvar()/][elseif (act.oclIsKindOf(CallOperationAction))]
[act.oclAsType(CallOperationAction).callOpe()/][elseif (act.oclIsKindOf(AddVariableValueAction))]
[act.oclAsType(AddVariableValueAction).genAddvar()/][elseif (act.oclIsKindOf(ReadStructuralFeatureAction))]
[act.oclAsType(ReadStructuralFeatureAction).genReadstr()/][elseif(act.oclIsTypeOf(ReadSelfAction))]
[act.oclAsType(ReadSelfAction).genReadSelf()/][else]
[act.name/]([for (it : String | act.input.invalue().genInput())separator (',')][it/][/for])[/if]
[/template]
[template  callOpe(ope : CallOperationAction)]
[if(ope.operation.class.ancestors(Package).name->includes('CollectionClasses') and ope.operation.name.matches('size'))]
[ope.target.invalue().genInput()/].Count[else]
[ope.target.invalue().genInput()/].[ope.operation.genOpe()/]([for (it : String | ope.argument.invalue().genInput())separator (',')][it/][/for])[/if]
[/template]
[template public genOpe(ope : Operation)]
[if (ope.class.ancestors(Package).name->includes('Alf'))]
[if (ope.class.ancestors(Package)->reject(oclIsTypeOf(Model)).name->includes('CollectionClasses'))]
[if (ope.name.matches('at'))]
[elseif(ope.name.matches('replaceAt'))]
set[elseif(ope.name.matches('add'))]
Add[else]
[ope.name/][/if][else]
[ope.name/][/if][else]
[ope.name/][/if]
[/template]
[template public callBeh(act : CallBehaviorAction)]
[if (act.behavior.ancestors(Package).name->includes('FoundationalModelLibrary'))]
[if (act.behavior.ancestors(Package)->reject(oclIsTypeOf(Model)).name->includes('BasicInputOutput'))]
[if (act.behavior.name='WriteLine')]
Console.WriteLine([act.argument.invalue().genInput()/])
[elseif (act.behavior.name='ReadLine')]
[let pin  : Pin = act.result->any(oclIsKindOf(Pin))]
Console.ReadLine()[/let][/if][elseif (act.behavior.ancestors(Package)->reject(oclIsTypeOf(Model)).name->includes('PrimitiveBehaviors'))]
[if (act.argument->size()=2)]
[act.argument->any(name='x').invalue().genInput()/][act.behavior.name/][act.argument->any(name='y').invalue().genInput()/][else]
[act.behavior.name/]([for (it : String | act.argument.invalue().genInput()) separator (',')][it/][/for][/if][/if][/if]
[/template]
[template public valuespe(act : ValueSpecificationAction)]
[if(act.value.oclIsTypeOf(LiteralString))]
"[act.value.stringValue()/]"[elseif (act.value.oclIsTypeOf(LiteralInteger))]
[act.value.integerValue()/][elseif (act.value.oclIsTypeOf(LiteralBoolean))]
[act.value.booleanValue()/][elseif (act.value.oclIsTypeOf(LiteralReal))]
[act.value.realValue()/][/if]
[/template]
[template public genReadSelf(act : ReadSelfAction)]
this
[/template]
```

Figure 12. A portion of the template for C#.

TABLE III. COMPARISON OF MODEL NODES AND GENERATED LINES.

Number of model nodes	Languages	Number of added or modified lines	Number of finished lines
94	Java	3	74
	C#	5	65

TABLE IV. COMPARISON OF DEVELOPMENT COSTS.

Languages	Production rate	New development cost	Development cost by reusing
Java	96%	131%	4%
C#	92%	152%	8%

diagrams. The source code of Java and C# was generated automatically. The same models as described by the previous processes were used for the model transformation of both languages. Operations were checked by evaluating the source code. Figures 6 and 7 show the class diagram and the behavior model of the system, respectively.

The generated source code in Java and C# for Class1 is shown in Figures 8 and 9, respectively. In addition, Fig. 10 shows the behavior model of Sortcommand, and Figures 11 and 12 show portions of the templates for Java and C#. The generated source code in Java and C# for Sortcommand is shown in Figures 13 and 14, respectively.

The development cost is evaluated according to [16]. It assumes that workload to add one node in UML diagrams equals that to describe one line of source code. Table III shows the number of model nodes, the number of lines of added or modified lines, and finished source code for each language. The added and modified lines correspond to parts that cannot be expressed by executable UML such as package, import, and so on. Table IV shows the rate of automatically generated code to the finished code. In addition, it shows the rate of cost of new development and reuse as compared with manual procedures starting from scratch to completion. The calculation formulas used in Table IV are shown below.

$$\text{Production rate} = (\text{finished code lines} - \text{added and modified lines}) * 100 / \text{finished code lines} \quad (1)$$

$$\text{New development cost rate} = (\text{model nodes} + \text{added and modified lines}) * 100 / \text{finished code lines} \quad (2)$$

$$\text{Reuse cost rate} = (\text{added and modified lines}) * 100 / \text{finished code lines} \quad (3)$$

Cost of the proposed method is about 130 –160% in developing new software, but it is reduced to less than 10% if reusing the models. According to the investigative report of IPA, ~70% of software development is reuse, modification, and migration of existing systems and new software development is ~30%. If a system is developed by the proposed method, the cost is

$$10*0.7+160*0.3=55\% \quad (4)$$

Although the proposed method is more expensive than manual procedures in new development, it can be less expensive when reusing the model(s) created for a system. Previously-created templates can be used in other projects and the cost declines further by repeating reuse. In the present software development environment, where reuse is common, a large cost reduction can be expected. Systems can be hierarchically divided into several (reusable) classes for every function.

```
1  package Package1;
2  import java.util.List;
3  public class Sortcommand {
4      public Sortcommand() {
5          super();
6      }
7      public void sortData(List<Integer> data) {
8          int dumy = 0;
9          int min = 0;
10         int j = 0;
11         int s = 0;
12         int k = 0;
13         while (k < data.size() == true) {
14             min = (int) data.get(k);
15             s = k;
16             j = k + 1;
17             while (j < data.size() == true) {
18                 if (min < (int) data.get(j) == true) {
19                     min = (int) data.get(j);
20                     s = j;
21                 } else
22                 if (min < (int) data.get(j) == false) {
23                 }
24                 j = j + 1;
25             }
26             dumy = (int) data.get(k);
27             data.set(k, data.get(s));
28             data.set(s,dumy);
29             k = k + 1;
30         }
31     }
32 }
```

Figure 13. Generated Java code of Sortcommand.

```
using System;
using System.Collections.Generic;
namespace Package1
{
public class Sortcommand {
    public  void sortData(List data) {
        int dumy=0;
        int min=0;
        int j=0;
        int s=0;
        int k=0;
        while(k<data.Count==true){
        min=data[k];
        s=k;
        j=k+1;
        while(j<data.Count==true){
        if(min<(data[j])==true){
        min=data[j];
        s=j;
        }else
        if(min<(data[j])==false){
        }
        j=j+1;
        }
        dumy=data[k];
        data.set(k,data[s]);
        data.set(s,dumy);
        k=k+1;
        }
    }
}
}
```

Figure 14. Generated C# code of Sortcommand.

V. Conclusion

Based on the trend where the rate of reuse, modification, and migration of existing systems is increasing in software development, an MDA method that uses executable UML jointly with class diagrams was proposed in this paper. The key idea and objective of the proposed method are to automatically generate source code that skeleton code does not have. As the result, the proposed method associates class operations with executable UML. Source code in Java and C# was generated from system models, and development costs were evaluated.

If the platform of a system is changed in the future, software developers cannot reuse existing source code, but they can reuse UML models to automatically generate source code in the new programming language. As a result, the proposed method can significantly reduce costs when models are reused. The proposed method can transform models into source code written in any type of programming language if there is an appropriate template. However, the method cannot correspond to a large scale of activity diagrams that contain a lot of classes and methods.

As future work, we believe it will be necessary to decide on a standard of model partitioning and a notation system for objects. In addition, an important future task will be to investigate what types of problems will occur when models are changed.

Acknowledgment

This work was supported in part by JSPS KAKENHI Grant Number JP16K06424.

References

[1] K. Matsumoto, K. Nakoshi, and N. Mori, “Intelligent software development method by model driven architecture,” Proc. of the Sixth International Conference on Intelligent Systems and Applications (INTELLI 2017), IARIA, July 2017, pp. 7-12, Nice, France.

[2] Information-Technology Promotion Agency, “Actual condition survey on software industry in the 2011 fiscal year,” (in Japanese) [Online]. Available from: https://www.ipa.go.jp/files/000004629.pdf, 2018.05.22.

[3] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, MDA distilled: Principle of model driven architecture. Addison-Wesley Longman Publishing Co., Inc. Redwood City, CA, 2004.

[4] T. Buchmann and A. Rimer, “Unifying modeling and programming with ALF,” Proc. of the Second International Conference on Advances and Trends in Software Engineering, pp. 10-15, 2016.

[5] M Usman, N. Aamer, and T. H. Kim, “UJECTOR: A tool for executable code generation from UML models. In Advanced Software Engineering and Its Applications, ASEA 2008, pp. 165-170, 2008.

[6] Papyrus User Guide, [Online]. Available from: http://wiki.eclipse.org/Papyrus_User_Guide, 2018.05.22.

[7] Object Management Group, “Semantics of a foundational subset for executable UML models,” [Online]. Available from: http://www.omg.org/spec/FUML/1.1/, 2018.05.22.

[8] Object Management Group, “List of executable UML tools,” [Online]. Available from: http://modeling-languages.com/list-of-executable-uml-tools/, 2018.05.22.

[9] Object Management Group, “Unified modeling language superstructure specification V2.1.2,” 2007.

[10] Acceleo: [Online]. Available from: http://www.eclipse.org/acceleo/, 2018.05.22.

[11] F. Budinsky, Eclipse modeling framework: A developer's guide. Addison-Wesley Professional, 2004.

[12] Object Management Group, “Metaobject facility,” [Online]. Available from: http://www.omg.org/mof/, 2018.05.22.

[13] Acceleo template: [Online]. Available from: https://wiki.eclipse.org/Acceleo/User_Guide#Extract_as_Template, 2018.05.22.

[14] A. Lanusse et al. “Papyrus UML: An open source toolset for MDA,” Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications, pp. 1-4, 2009.

[15] UML to Java Generator [Online]. Available from: https://marketplace.eclipse.org/content/uml-java-generator, 2018.05. 22.

[16] K. Matsumoto, T. Maruo, M. Murakami, and N. Mori, “A graphical development method for multiagent simulators,” Modeling, Simulation and Optimization - Focus on Applications, Shkelzen Cakaj, Eds., pp. 147-157, INTECH, 2010.