



Proposal of a method to support testing for Java programs with UML

メタデータ	言語: English 出版者: IEEE Computer Society 公開日: 2010-02-25 キーワード: software testing, Unified Modeling Language (UML), test items, functional testing, Java 作成者: 片山, 徹郎, Yabuya, Yusuke メールアドレス: 所属:
URL	http://hdl.handle.net/10458/2563

Proposal of a Method to Support Testing for Java Programs with UML

Tetsuro Katayama Yusuke Yabuya
Department of Computer Science and Systems Engineering,
Faculty of Engineering, University of Miyazaki.
1-1 Gakuen-kibanadai nishi, Miyazaki 889-2192, Japan.
kat@cs.miyazaki-u.ac.jp, yuu@earth.cs.miyazaki-u.ac.jp

Abstract

There is an increasing need for effective testing of software for complex safety-critical applications. This paper proposes a supporting method of testing for Java programs by using Unified Modeling Language (UML) in order to improve the reliability of them. The correspondences of source codes in Java and elements of a class diagram, a sequence diagram, and a statechart diagram have been extracted. By using the extracted correspondences as test items in testing, it becomes possible to test effectively the specification of software, the structure of Java programs, the flows of processing of a system, and the flow of transition between states. Moreover, in order to confirm the validity of the proposed method, a prototype tool to support testing for Java programs is implemented. The inputs of this prototype are three diagrams, and the outputs are test items generated from the extracted correspondences. As an example, a source code of blackjack game in Java has been tested by using the test items outputted from the prototype. It has been possible to test 67.2% of the number of lines in the whole source code.

keywords: software testing, Unified Modeling Language (UML), test items, functional testing, Java.

1 Introduction

There is an increasing need for effective testing of software for complex safety-critical applications, such as avionics, medical, and other control systems. This paper proposes a supporting method of testing for Java programs by using Unified Modeling Language (UML)[1, 2], in order to improve the reliability of programs written in Java programming language[3].

UML has been proposed to OMG (Object Management Group)[4] as a unification methodology, and agreed officially with UML1.0 in 1997. After that, UML is expanded and it is UML2.0 in Oct., 2004. UML has been accepted as an industrial standard for specifying, visualizing, understanding, and documenting object-oriented software systems. The object-oriented language repre-

sented by Java and C++ is spreading quickly. For programming with an object-oriented language, an object-oriented analysis and design is required. After Smalltalk called first object-oriented language in 1970 is proposed, various object-oriented methodology is advocated in about 35 years. UML is becoming leading now.

In this paper, correspondences of source codes in Java and elements of a class diagram, a sequence diagram, and a statechart diagram, which are UML diagrams, are extracted. The correspondences are extracted from the specifications of UML and Java. By using the extracted correspondences as test items in testing, it becomes possible to test effectively the specification of software, the structure of Java programs, the flows of processing of a system, and the flow of transition between states.

Moreover, in order to confirm the validity of the proposed method, a prototype tool to support testing for Java programs is implemented. The inputs of this prototype are a class diagram, a sequence diagram, or a statechart diagram drawn by using GUI (Graphical User Interface), and the outputs are test items generated based on the correspondences extracted in the proposed method.

Section 2 shows the correspondences extracted in this paper. Section 3 describes a prototype tool to support testing for Java programs to confirm the validity of the proposed method in Section 2. It is implemented in Java and is applying to an example. Section 4 describes discussion and evaluation.

2 Correspondences of source codes in Java and UML diagrams

This paper proposes a supporting method of testing for Java programs by using UML, in order to improve the reliability of Java programs. In order to realize this method, correspondences of source codes in Java and elements of UML diagrams are extracted. This section explains the correspondences. A class diagram, a sequence diagram, and a statechart diagram are adopted in this paper.

2.1 Class diagram

2.1.1 Attributes, operations, and visibility

Attributes in a class diagram correspond to variables. For this reason, the variables defined in the class diagram must be declared in a source code. Operations correspond to methods. Hence, the methods defined in the class diagram must be declared in a source code. On the contrary, variables or methods which are not defined in the class diagram must not be declared in a source code. As an exception, even if only a method of a class name (initialization method) does not exist in a class diagram, it can be declared in a source code. As to visibilities, their abbreviations are used before an attribute, operation, and a roll name, and it means how it is visible from other objects. In Java, + is public, # is protected, and - is private.

Figure 1 shows a class diagram of class Person and its source code implemented in Java. #name:String is defined as an attribute in the class diagram. It is ascertained that its visibility is protected, its type is String, and its name is name. Hence, in a source code in Java,

```
protected String name
```

must be declared in class Person.

As to an operation, +getName():String is defined an operation in the class diagram. It is ascertained that its visibility is public, its type is String, and its name is getName. Hence, in a source code in Java,

```
public String getName()
```

must be declared in class Person.

The method public Person(String name) which is not defined in the class diagram exists in its source code. Since it is an initialization method, it is not an error.

2.1.2 Associations

If an association exists between classes, in Java, an instance of a class with the association is stored in an instance variable, and is possessed. Otherwise, an instance of other classes is not stored in an instance variable, and is not possessed. When a direction is given to an association, it will become the meaning “unidirectional navigatable association”, and the instance of the only class of the arrowhead side is stored in an instance variable, and is possessed.

Figure 2 shows an example of a bidirectional navigatable association between two classes exists. In class A, the instance variable b in class B is defined. Hence, in a source code in Java,

```
private B b;
```

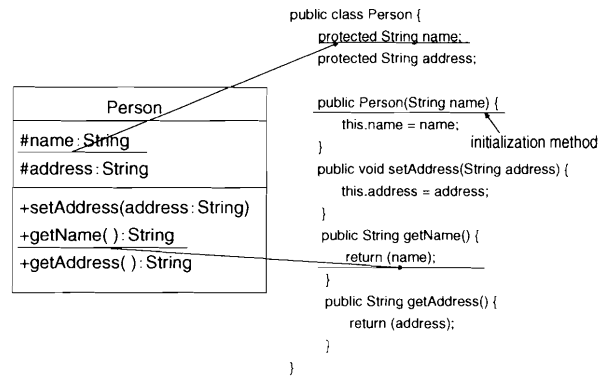


Figure 1. A class diagram of Person and its source code

must be declared in class A. In class B, the instance variable a in class A is defined. Hence, in a source code in Java,

```
private A a;
```

must be declared in class B.

Figure 3 shows an example of a unidirectional navigatable association between two classes exists. In class A, as similar to Figure 2, the instance variable b in class B is defined. Hence, in a source code in Java,

```
private B b;
```

must be declared in class A. Class B does not need to define an instance variable.

2.1.3 Aggregations and Compositions

If an aggregation exists, variables of the class equivalent to the whole are implemented as object variables of the class equivalent to a portion. Figure 4 shows an example of aggregations. In a source code in Java,

```
private Person owner_;
private Person account_;
private Vector employee_ = new Vector();
```

must be declared in class Company.

As to compositions, aggregations and compositions cannot be distinguished in the specification of Java. For this reason, when compositions exist, in a source code in Java, it is defined definition as similar to an aggregation.

2.1.4 Generalizations

If a generalization exists, the class of the arrowhead side is inherited. In a source code in Java, it corresponds using delimiter extends.

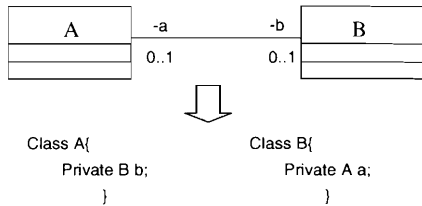


Figure 2. An example of a bidirectional navigable association

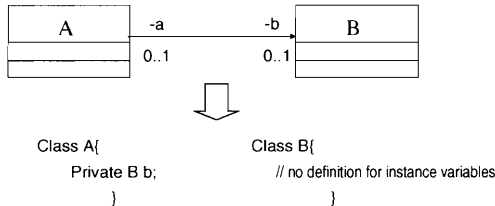


Figure 3. An example of a unidirectional navigable association

Inheritance is succeeding attributes and operations in the class which have already been prepared and is creating a new class. Hence, when a generalization exists, the sub-classes can use the variables and methods declared in the super-class, unless their visibilities are **private**. It can use only in the class itself in which variables exist when their visibilities are **private**.

Figure 5 shows an example of a generalization. Class Customer has inherited class AbstractPerson. Hence, class Customer must be declared as

```
public class Customer extends AbstractPerson
```

in a source code in Java.

Although a variable **name** exists in the source code in Figure 5, it does not exist in the attributes of class Customer. However, the variable **name** is not an error since it is a variable in class AbstractPerson, its visibility is **protected**, and class Customer is a sub-class of class AbstractPerson. Hence, in this case, it can be used.

2.1.5 Multiplicity

Since a multiplicity means how many a class has associations between the class and instances, in case a multiplicity is zero or one, a source code are needed to write as similar to associations. When a multiplicity is plurality (0-N or 1-N), it needs to realize an array of class type whose multiplicity is plurality, or to realize an array of **Vector** type in a source code in Java. When a multiplicity is constant, it is necessary to realize an array of

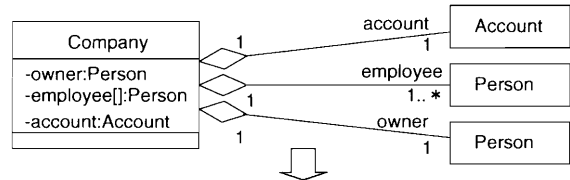


Figure 4. An example of aggregations

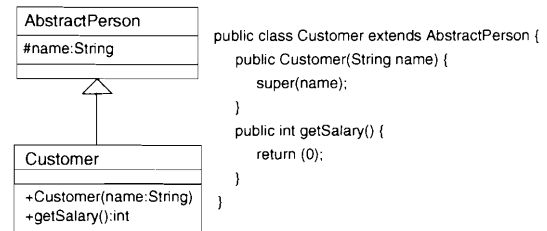


Figure 5. An example of a generalization

the maximum value of the multiplicity. In Java, in order to create an array, it is necessary to use operator **new**. As to the minimum value of a multiplicity, it cannot be declared in the specification of Java.

Figure 6 shows an example in case a multiplicity is plurality. It means that class A can possess 0-N instances of class B. It is necessary to use operator **new** to create an array. Hence, in a source code in Java,

```
private B[] b = new B[];
```

must be declared in class A. Or in a source code in the case of using an array of **Vector** type,

```
private Vector b = new Vector();
```

must be declared in class A. When a multiplicity is constant, the maximum value of the multiplicity will be put in [].

2.1.6 Interfaces

If an interface exists, it can be corresponded to delimiter **implements** which means implementation of an interface. Figure 7 shows an example of interface **IPerson** which class **AbstractPerson** refines. Hence, class **AbstractPerson** must be declared as

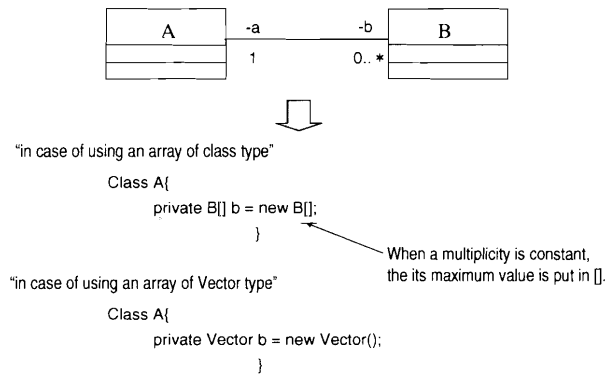


Figure 6. An example in case a multiplicity is plurality.

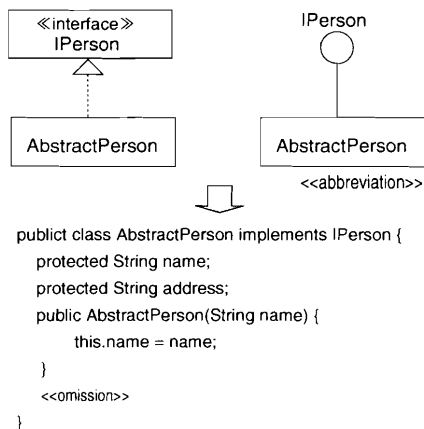


Figure 7. An example of interfaces

public class Customer extends AbstractPerson

in a source code in Java.

The above is the extracted correspondences of elements of a class diagram and source codes in Java. Table 1 summarized the correspondences.

2.2 Sequence diagram

When an input arrow exists in an object lifeline which is an element of a sequence diagram, it is necessary to declare a method for a label of the arrow inside the class of the object in Java. But, since a class cannot be specified when the class name of an object symbol is hidden, it is impossible to declare the method inside a specific class. When an output arrow exists in an object lifeline, it is necessary to declare a method call for a label of the arrow inside the method outputting the arrow in Java.

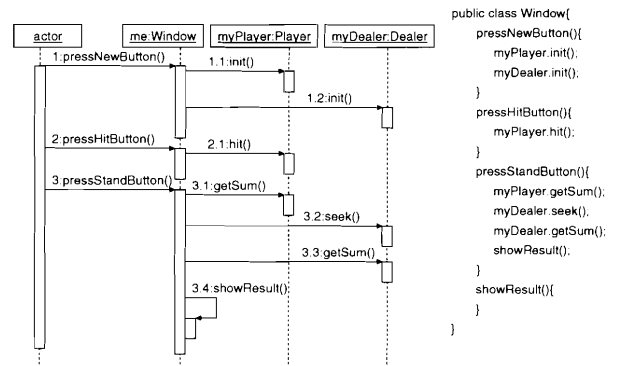


Figure 8. A sequence diagram of blackjack and its source code.

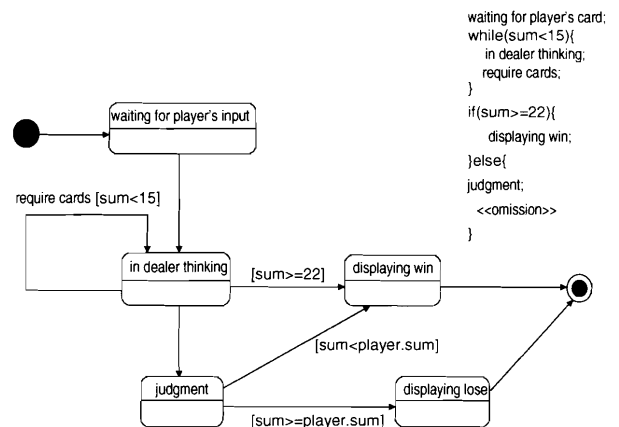


Figure 9. A statechart diagram of class dealer of blackjack and its source code.

Figure 8 shows a sequence diagram of blackjack, which is a famous card game, and a source code in Java of class Window. First, since pressNewButton inside class Window is called from actor, it is necessary to declare method pressNewButton inside class Window in a source code in Java. Its call cannot be declared since actor is not a class name.

And then, function pressNewButton calls function init in class Player and Dealer. Hence, it is necessary to declare the call of function init of object myPlayer and myDealer inside method pressNewButton in a source code in Java. Simultaneously, function init must be declared inside class Player and Dealer.

The above is the extracted correspondences of elements of a sequence diagram and source codes in Java. Table 2 summarized the correspondences.

Table 1. Correspondences of elements of a class diagram and source codes in Java.

an element in class diagram	correspondences to source codes in Java
attribute	It is declared as a variable. In a class diagram, "visibility attribute_name:type = initialized_value" is described. In a source code in Java, "visibility type attribute_name = initialized_value" must be written.
operation	It is declared as a method. In a class diagram, "visibility operation_name:type_of_return_value" is described. In a source code in Java, "visibility type operation_name" must be written.
visibility	Its abbreviations are used before an attribute, operation, and a roll name, and it means how it is visible from other objects. In Java, + is public , # is protected , and - is private .
association	An instance of a class with the association is stored in an instance variable, and is possessed. In case of a unidirectional navigatable association, the instance of the only class of the arrowhead side is stored in an instance variable, and is possessed.
aggregation	Variables of the class equivalent to the whole are implemented as object variables of the class equivalent to a portion.
composition	In the specification of Java, aggregations and compositions cannot be distinguished. For this reason, it is defined definition as similar to an aggregation.
generalization	The class of the arrowhead side is inherited. In a source code in Java, it corresponds using delimiter extends . If class A inherits class B, class A is declared as public class A extends B . The sub-classes can use the variables and methods declared in the super-class, unless their visibilities are private .
multiplicity	In case of zero or one, a source code is needed to write as similar to associations. In case of plurality (0-N or 1-N), a multiplicity needs to realize an array of class type whose multiplicity is plurality, or it is necessary to realize an array of Vector type. In case of constant, it is necessary to realize an array of the maximum value of the multiplicity. In Java, in order to create an array, operator new is used.
interface	If class A implements interface B, class A must be declared as public class A extends B .

Table 2. Correspondences of elements of a sequence diagram and source codes in Java.

an element in sequence diagram	correspondences to source codes in Java
In case of an input arrow existing in an object lifeline	It is necessary to declare a method for a label of the arrow inside the class of the object in Java. But, since a class cannot be specified when the class name of an object symbol is hidden, it is impossible to declare the method inside a specific class.
In case of an output arrow existing in an object lifeline	It is necessary to declare a method call for a label of the arrow inside the method outputting the arrow in Java.

2.3 Statechart Diagram

When a loop exists in a statechart diagram, it is necessary to implement it by using statements having repetitive structure (e.g. **while** or **for** statement) in Java. In this case, its guard conditions correspond to the conditional expressions of the statements having repetitive structure. When a guard condition exist in an output transition or multiple output transitions exist, it is necessary to implement it by using statements having selection structure (e.g. **if-else** or **switch** statement) in Java. In this case, its guard conditions correspond to the conditional expressions of the statements having selection structure.

Moreover, when multiple output transitions exist and a loop exists, it is necessary to implement the statements having selection structure after the statements having repetitive structure. The things as above inside the class which the statechart diagram expresses must

be implemented.

Figure 9 shows a statechart diagram of class Dealer of blackjack and its source code in Java. The state "in dealer thinking" has three output transitions, and one among them is a loop with an event.

Suppose that **if-else** statement uses as a statement having selection structure and **while** statement uses as a statement having repetitive structure. First, since the guard condition of the loop is **sum<15** it is necessary to describe **while (sum<15)** in a source code in Java. Next, since a guard condition is **sum>=22** in another transition to state "displaying win" and a guard condition does not exist in the other transition to state "judgment", it is necessary to describe **if (sum>=22)** to state "displaying win" and then to use **else** statement to the transition to state "judgment".

The above is the extracted correspondences of elements of a statechart diagram and source codes in Java. Table 3 summarized the correspondences.

Table 3. Correspondences of elements of a statechart diagram and source codes in Java.

an element in state diagram	correspondences to source codes in Java
In case of a loop existing	It is necessary to implement it by using statements having repetitive structure (while or for statement). Its guard conditions correspond to the conditional expressions of the statements.
In case of a guard condition existing in an output transition or multiple output transitions existing	It is necessary to implement it by using statements having selection structure (if-else or switch statement). Its guard conditions correspond to the conditional expressions of the statements.
In case of multiple output transitions and a loop existing	It is necessary to implement the statements having selection structure after the statements having repetitive structure.

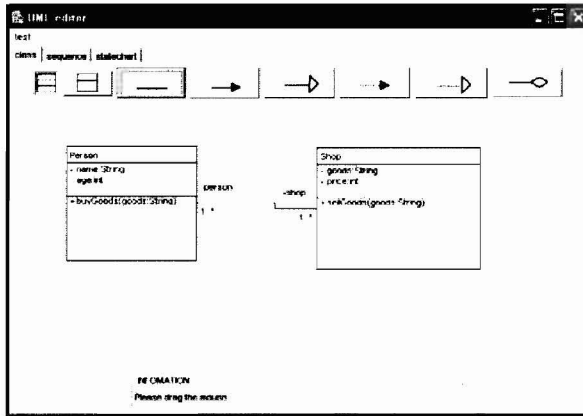


Figure 10. a class diagram including an association (drawn on the input window of the prototype).

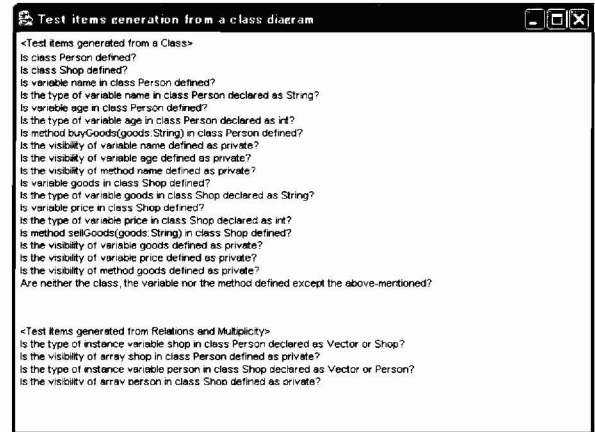


Figure 11. Test items generated the class diagram from Figure 10. (Output of the prototype.)

3 A prototype tool to support testing for Java programs

This paper proposes a supporting method of testing for Java programs by using UML. The correspondences of source codes in Java and elements of a class diagram, a sequence diagram, and a statechart diagram are extracted in Section 2. In this section, the validity of the proposal method is confirmed by implementing a prototype tool to support testing for Java programs. This prototype has been implemented in Java[3] so that it can be executed on different operating systems.

The inputs and outputs of the implemented prototype is described. The inputs of the prototype are a class diagram, a sequence diagram, or a statechart diagram drawn by using GUI (Graphical User Interface), and the outputs are test items generated from on the correspondences extracted in the proposed method.

The procedure of the prototype is described. First, either of three diagrams of a class diagram, a sequence

diagram, and a statechart diagram is chosen with the tab and is drawn on an input window. And then, clicking the “test” in the upper left of the input window displays an output window. In the output window, test items generated automatically are displayed by itemized statement based on the correspondences extracted in this paper from diagrams drawn on the input window.

Figure 10 shows a class diagram, which includes two classes Person and Shop, and an association between the classes, drawn on the input window of the prototype implemented in this paper. The test items generated from this figure. Figure 11 shows the output window obtained by clicking “test” in the upper left of Figure 10.

Test items are obtained from Figure 11. The sentences currently displayed on <test items generated from a Class> are test items extracted from two classes Person and Shop, and four sentences currently displayed on <test items generated from Relations and Multiplicity> are test items extracted from the association between

Table 4. The rate in which we can test a source code (with the number of lines).

source code	the number of lines tested from a class diagram	the number of lines tested from a sequence diagram	the number of lines tested from a statechart diagram	the number of lines which we can test	the number of lines of the whole source code	The rate in which we can test the source code
class Parent	29	40	28	83	112	74.1 %
class Child	22	23	6	37	52	71.2 %
class Player	15	13	14	31	48	64.6 %
class Card	17	8	12	29	47	61.7 %
class Cardstack	11	13	7	22	31	71.0 %
class Blackjack	16	27	15	48	82	58.5 %
Total	110	124	82	250	372	67.2 %

two classes.

It has confirmed that the test items can be outputted correctly based on the correspondences extracted in this paper about attributes, operations, visibility, associations, and multiplicity as elements of a class diagram. Moreover, it has confirmed that the test items can be outputted correctly based on the correspondences as to the other elements of a class diagram. Similarly, it has confirmed as to elements of a sequence diagram and a statechart diagram.

Here, time until an output window is displayed from each UML diagram drawn on an input window time takes about 1 second. The prototype has been executed on CPU: Pentium4 1.80GHz, Memory:512MB, and OS:Windows XP.

4 Discussion and evaluation

In this paper, a supporting method of testing for Java programs by using UML[1, 2] has proposed in order to improve the reliability of programs written in Java programming language[3]. The correspondences of source codes in Java and elements of a class diagram, a sequence diagram, and a statechart diagram have been extracted. By using the extracted correspondences as test items in testing, it becomes possible to test effectively the specification of software, the structure of Java programs, the flows of processing of a system, and the flow of transition between states.

In order to confirm the validity of the proposed method, a prototype tool to support testing for Java programs is implemented. Table 4 shows the result investigated the rate in which we can test a source code of blackjack in Java by using the test items outputted from the prototype implemented in this paper. It is based on the number of lines.

From Table 4, we can conclude that any source code can be tested in range about 60 to 70 percent. It is possible to test 67.2% in the whole source code in this

example. Here, when the number of lines which we can test from each diagram is compared with the total number of lines, the rate in the total number of lines is lower. It is because lines which we can test in each diagram overlap.

Consequently, the improvement in reliability of programs written in Java is expected by using the proposed method. The rate has been measured with the number of lines of a source code for convenience in this time. If structural testing with coverages is used together, more effective testing may be expectable.

In this paper, the correspondences have been extracted from a class diagram, a sequence diagram, and a statechart diagram. By extracting correspondences between the other diagrams of UML and in a source code in Java from now on, test items increase, and the rate in which we can test the whole improves, and then it becomes possible to test it more effectively. Moreover, it may also become possible to check adjustment between UML diagrams by using the test items which each UML diagram overlaps[5].

In recent years, MDA(Model Driven Architecture)[6] which develops a system focusing on its model expressed by diagrams attracts attention. Software tools which actually generate a template of a source code in Java automatically from UML diagrams are marketed[7, 8]. Such tools have not only a source code automatic generation function but also a reverse conversion function (reverse engineering) which generates a model from a source code.

If a source code automatic generation tool is applied in software development, the prototype to support testing implemented in this paper this time may become unnecessary. However, a template of a source code generated automatically by the generation tools must add to be written by a hand and then consequently to test the source code after adding to write.

On the other hand, a main purpose of a reverse conversion function is maintenance of a source code. Re-

turning the original diagrams is not taken into consideration. For this reason, as diagrams become complicated, all elements of the diagrams from a source code cannot be generated as the original diagrams. That is, it is difficult to test whether a source code is implemented as UML diagrams by using the reverse conversion function.

Moreover, when a UML diagram used as an input is mistaken, a mistaken source code may be generated. In the proposed method, when errors are noticed in a source code, not only the source code but also errors of UML diagrams may be able to be noticed by referring to its specification.

Various researches for a UML-based testing method have already started. [9, 10, 11] have proposed coverage criteria to generate test data based on events (transitions) of a statechart diagram or a collaboration diagram. They are methods for a structural testing and are important researches. They differ from our approach, which generates test items from all elements of diagrams, for a functional testing. [12] has suggested a testing method to derive test requirements from a class diagram and a collaboration diagram. It introduces a scripting language over a concept of UML in order to derive accurate test requirements. In the proposed method, the correspondences have extracted from only the specification of UML. Anybody who learns only UML is available and understandable for the method.

5 Conclusion

This paper has proposed a supporting method of testing for Java programs by using UML in order to improve the reliability of them. The correspondences of source codes in Java and elements of a class diagram, a sequence diagram, and a statechart diagram, which are UML diagrams, have been extracted. By using the extracted correspondences as test items in testing, it becomes possible to test effectively the specification of software, the structure of Java programs, the flows of processing of a system, and the flow of transition between states.

Moreover, in order to confirm the validity of the proposed method, a prototype tool to support testing for Java programs is implemented. The inputs of this prototype are a class diagram, a sequence diagram, or a statechart diagram drawn by using GUI, and the outputs are test items generated from the extracted correspondences.

As an example, we have tested a source code of black-jack game in Java by using the test items outputted from the prototype implemented in this paper. It has been possible to test 67.2% of the number of lines in the whole source code. Consequently, the improvement in reliability of programs written in Java is expected by using the proposed method.

Future issues are as follows:

- More extraction of the correspondences.

It is possible to extract more correspondences. In this paper, the correspondences have been extracted from a class diagram, a sequence diagram, and a statechart diagram. By extracting correspondences between the other diagrams of UML and in a source code in Java from now on, test items increase, and the rate in which we can test the whole improves, and then it becomes possible to test it more effectively.

- Automation of the proposed method.

At present, it is tested whether any error exists in a source code by comparing the test items generated by the prototype and the source code in Java. In future, automation of the test which used the outputted test items must be considered.

References

- [1] J. Rumbaugh, I. Jacobson, and G. Booch: "The Unified Modeling Language Reference Manual," Addison Wesley Publishing Company (1999).
- [2] UML Home page: <http://www.uml.org/>
- [3] D. Flanagan: "Java in a Nutshell, 3rd Edition," O'Reilly & Associates (1999).
- [4] Object Management Group: <http://www.omg.org/>
- [5] T. Katayama: "Proposal of a Supporting Method for Diagrams Generation with the Transformation Rules in UML," *Proc. 2002 Asia-Pacific Softw. Eng. Conf. (APSEC 2002)*, pp.475-484 (2002).
- [6] D. S. Frankel: "Model Driven Architecture: Applying Mda to Enterprise Computing," John Wiley & Sons (2003).
- [7] IBM: "Rational Rose," <http://www-306.ibm.com/software/rational/>
- [8] Embarcadero Technologies, Inc.: "Describe," <http://www.embarcadero.com/>
- [9] J. Offutt and A. Abdurazik: "Generating Tests from UML Specifications," *Proc. 2nd Int'l Conf. on the UML (UML'99)*, pp.416-429 (1999).
- [10] A. Abdurazik and J. Offutt: "Using UML collaboration diagrams for static checking and test generation," *Proc. 3rd Int'l Conf. on the UML (UML'00)*, pp.383-395 (2000).
- [11] Y. Wu, M.-H. Chen, and J. Offutt: "UML-based Integration Testing for Component-based Software," *Proc. 2nd Int'l Conf. on COTS-Based Softw. Sys. (ICBSS)*, pp.251-260 (2003).
- [12] L. Briand and Y. Labiche: "A UML-based approach to system testing," *Proc. 4th Int'l Conf. on the UML (UML'01)*, pp.194-208 (2001).