TFVIS: a Supporting Debugging Tool for Java Programs by Visualizing Data Transitions and Execution Flows

Hiroto Nakamura[*], Tetsuro Katayama[*] ,Yoshihiro Kita[†]
Hisaaki Yamaba[*], Kentaro Aburada[‡], Naonobu Okazaki[*]
[*]*University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
[†]*Kanagawa Institute of Technology, 1030 Shimo-ogino, Kanagawa, 243-0292 Japan*
[‡]*Oita National College of Technology, 1666 Maki, Oita, 870-0152 Japan*
E-mail: tf13006@student.miyazaki-u.ac.jp, kat@cs.miyazaki-u.ac.jp, kita@earth.cs.miyazaki-u.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@oita-ct.ac.jp, oka@cs.miyazaki-u.ac.jp

Abstract

TFVIS(transitions and flow visualization) can perform the visualization of data transitions and the visualization of execution flows. The visualization of data transitions shows the flow of variable renewals in executing programs. It becomes easier to grasp the behavior in executing the programs whose behavior is unexpected by a bug. The visualization of execution flows shows an entire flow of the execution. It is useful to select the part where users want the visualization of the data transitions.

Keywords: Debugging, Java, Dynamic analysis, Visualization, Slicing

1. Introduction

We need much time in order to find the cause of a bug in programs.[1] We need to find the behavior of programs which is different from ideal behavior in order to find the cause of a bug. However, grasping the behavior is difficult because the behavior of the program which includes a bug becomes unexpected behavior.

We had implemented the visualization tool called TVIS[2] in our previous research. TVIS visualizes the data transitions. The data transitions in our research show the flow of variable renewals in executing a program.

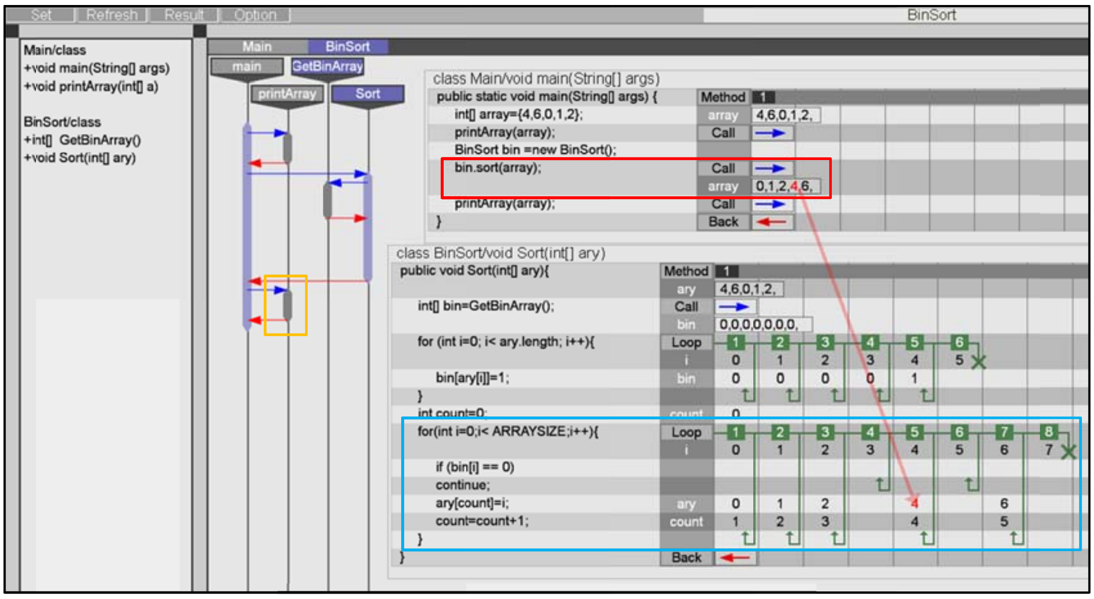
TVIS expresses when and what value of each variable is renewed. Programmers can grasp behavior of a program because they can prefigure the behavior of each variable at arbitrary timing in the program execution by grasping the data transitions.

However, the efficiency of TVIS is lost, if the program which TVIS uses becomes large. Moreover, TVIS can't visualize the data transitions between different methods.

Therefore, this paper implements the visualization tool TFVIS(transitions and flow visualization) which visualizes the data transitions and the execution flows, and shows the effectiveness of TFVIS. That is, we combine the visualization of the execution flows with the visualization of the data transitions in order to visualize a larger program. Moreover, we improve the visualization of the data transitions in order to realize the more effective one.

2. TFVIS

We have developed the visualization tool called TFVIS. TFVIS visualizes the data transitions and the execution flows of Java programs. Fig.1 shows an example of the window of TFVIS. Two small windows on the right of the window are called the data transitions diagrams. The diagram on the left-hand side of them is called the execution flows diagram.



```
Main/class
+void main(String[] args)
+void printArray(int[] a)

BinSort/class
+int[]  GetBinArray()
+void Sort(int[] ary)
```

```
public static void main(String[] args) {
    int[] array={4,6,0,1,2};
    printArray(array);
    BinSort bin =new BinSort();
    bin.sort(array);
    printArray(array);
}
```

```
public void Sort(int[] ary){
    int[] bin=GetBinArray();
    for (int i=0; i< ary.length; i++){
        bin[ary[i]]=1;
    }
    int count=0;
    for(int i=0;i< ARRAYSIZE;i++){
        if (bin[i] == 0)
        continue;
        ary[count]=i;
        count=count+1;
    }
}
```

```
array  4,6,0,1,2,
array  0,1,2,4,6,
ary    4,6,0,1,2,
bin    0,0,0,0,0,0,0,
```

Fig. 1. An example of the window of TFVIS.

The list of the left side of the window of TFVIS shows list of methods of a program. The functions of TFVIS are as follows.

2.1. *Program analysis*

Firstly, we explain the program analysis used to the visualization. TFVIS performs two types of analysis: the structure analysis and the executions analysis.

The structure analysis is static analysis for obtaining the information which is used to insert probes for the execution analysis and visualize. The information which is obtained by the structure analysis is the information about a location where statements occur: generations and renewals of variables, loops, and method calls. Moreover, it includes the information about the relation of a class and a method.

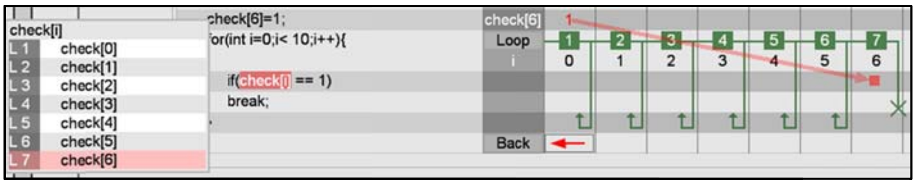
The execution analysis is dynamic analysis for obtaining the information about behavior in executing the programs. The information which is obtained by the execution analysis is the information about renewals of variables, behavior of loops and method calls, and so on. TFVIS obtains the information by using the probes, inserted based on the result of the structure information.

2.2. *Data transitions diagram*

The data transitions diagram is generated by the visualization of the data transitions. TFVIS in Fig.1 shows the two windows of the data transitions diagram. TFVIS can show the plural windows of the data transitions diagram in this way.

The data transitions diagram shows renewals of a variable. It is a table and indicates the number of iterations of execution in a lateral direction and indicates the line number of a source code in a vertical direction. Normally, its table shows renewal value of variables to where the renewal of the variable occurs. The area surrounded by the red frame in Fig.1 shows that the values of the array “array” become “0, 1, 2, 4, 6” after the method “sort” is executed. The method “printArray” has “array” as an argument the same as "sort". However, the renewal value of “printArray” is not showed, because “printArray” only refers to the value of "array" without updating it.

The green pattern on the data transitions diagram shows a loop, and the number on it shows the number of the iteration of the loop. The green arrows extending from the right of each number show the process of iteration of the loop. In the statement “for” is surrounded by the blue frame in Fig.1, the arrow at the fourth loop in this example is shorter than the arrow at the third loop; it expresses “jump” by using statement “continue”. The mark “x” on the seventh loop expresses the end of a loop caused by using the “break” statement or failing the condition of a loop.



```
check[6]=1;
for(int i=0;i< 10;i++){

  if(check[i] == 1)
  break;
```

Fig. 2. An example of the window of the data transitions search.

2.2.1. *Data transitions arrow*

The data transitions arrow is a function of the data transitions diagram, and visualizes the dependence relations between each renewal of variables. The dependence relations mean ones between state of a variable and the state of other variables which is used for the renewal of its state.

The data transitions arrow is based on the technique of the program slicing.[3] It becomes easy to grasp behavior of the data transitions in the loop and select the standard value of the program slicing by combining the technique of the program slicing and the data transitions diagram.

TFVIS shows the data transitions arrow, when users click a value of a variable on the data transitions diagram. The data transitions arrow is the red arrow and connects the two values which the data transitions diagram shows with red font color. Fig.1 shows an example of it. It can visualize the relation of the values between methods. Therefore, it is not necessary that the start point and the end point of the data transitions arrow exists in the same method.

Fig.1 shows the data transitions arrow in the window when users select the fourth value of the array "array" which the method "sort" updates. The tip of the data transitions arrow of Fig.1 shows the state which finally updated the value which the user selected. When users find the suspicious state of a variable, the data transitions arrow supports to find the cause of it.

2.2.2. *Data transitions search*

The data transitions search shows the data transitions arrow by selecting the variable and its state from the source code on the left side of the data transitions diagram. Fig.2 shows an example of the data transitions search, it is the window when users selected the state of the seventh loop of the variable "check[i]" which is used as the conditions of the "if" statement.

2.2.3. *Selection of the target for the visualization*

The data transitions diagram can't visualize the whole of the program at a time, because visualization typically has the issue[4] that the output of it becomes huge even if the visualized program is moderate size.

Therefore, users must select the part where they want the visualization of the data transitions. The execution flow diagram, which we explain in the next section, can support this selection.

2.3. *Execution flow diagram*

The visualization of the execution flow visualizes an entire flow of execution of a program and generates the execution flow diagram. The execution flow diagram provides the useful information in order to select the part where users want the visualization of the data transitions. It is based on the sequence diagram and shows status of use of each methods and the relation of method calls.

We explain the usage of the execution flow by using an example of Fig.1. The column in topmost of the execution flow diagram shows each class name, and the figures at under of it express each method. The black lines extending from methods are their lifelines, and the thick parts on their lifelines are their execution specification which means execution of the method. When users click a execution specification, TFVIS shows the data transitions diagram of corresponding with it. Normally, the execution specification uses the gray color. However, the execution specification uses the light blue color, if the window of the data transitions diagram of corresponding with it is being displayed. The blue arrows on the execution flow diagram express method calls, and the red arrows express completion of methods.

The execution specification is surrounded by the orange frame in Fig.1 expresses the execution which is called by the only execution of the method "main" and is the second execution of the method "printArray".

3. Discussion

We discuss the usefulness of TFVIS.

Programmers normally use the dynamic slicing[5] or trace tools[6,7] in order to obtain data transitions in executing a program. However, it is difficult to grasp positional relations of each renewal, if data transitions include the process of a loop, and so on. The visualization of data transitions of TFVIS uses the representation which exploits the source code and the processes of a loop as shown in Fig.1, and can show the data transitions in a way that is easy to understand. Moreover, the above-mentioned technique or tools need that users search a reference point when they analyze a program. In the case of TFVIS, it becomes easy to decide a reference point because the data transitions diagram shows renewals of variables and behavior of loops so that users understand them at a glance.

On the other hand, the range of the program which TFVIS can visualize is still narrower than other tools because the abstraction degree of the visualization of TFVIS is lower. TFVIS is inferior compared with the tool[8] which can visualize the multi-threading because TFVIS can't visualize multi-threading.

We will be enable to adjust the abstraction degree of the visualization of the execution flow in order to enlarge the range of the program which TFVIS can visualize. The abstraction degree of the execution flow diagram is too low to prevent excessive expansion of its diagram, if users want to visualize the program which has the process of multi-thread or a larger program. However, if it merely becomes higher, the users become unable to grasp behavior of a program in detail. Therefore, the information which the users obtain by the execution flow diagram becomes not enough to select the part where they want the visualization of the data transitions. We combine the visualization which the abstraction degree is low and the visualization which the abstraction degree is high by using an idea of the fisheye view[9]. Therefore, we will enable TFVIS to provide the enough information which users select the part where they want the visualization of the data transitions and visualize the larger programs.

4. Conclusion

We have implemented TFVIS in order to improve the efficiency of debugging of Java program.

TFVIS can support to grasp behavior in executing a program by visualizing the data transitions. The data transitions diagram visualizes renewals of variables and behavior of loops. Even if the data transition is the relation between methods, the data transitions arrow can visualize it.

Moreover, we have implemented the visualization of execution flow in order to improve the narrowness of the range of the visualization of the data transitions. It improves convenience when TFVIS visualizes a large program. Therefore, we judge that TFVIS can support to find the cause of a bug, and is effective of debugging for Java programs.

The future issues are as follows.

- The realization of the visualization of the multi-thread process.
- The improvement of the execution flow diagram by using an idea of the fisheye view.

References

1. Roger S. Pressman, Software Engineering A Practitioner's Approach 5thEdition, *McGraw-Hill Science* (2001).
2. Tetsuro Katayama et al, Proposal of a Visualizing Method of Data Transitions to Support Debugging for Java Programs, *Journal of Robotics Networks and Artificial Life*, **1**(2) (2011) 111-115.
3. Mark Weiser, Programmers Use Slices When Debugging,*Communications of the ACM*, **25** (1982) 446-452.
4. W. De Pauw et al, Execution patterns in object-oriented visualization, In *Proc. 4th COOTS* (1998) 219-234.
5. H Agrawal, JR Horgan, Dynamic Program Slicing, *SIGPLAN Notices* **25**(6) (1990) 246-256 .
6. Kouhei Sakurai et al, Traceglasses: A Trace-based Debugger for Realizing Efficient Navigation, *Information Processing Society of Japan*, **3**(3) (2010) 1-17.
7. Salman Mirghasemi et al, Querypoint : Moving Backwards on Wrong Values in the Buggy Execution, *ESEC/FSE* (2011) 436-439.
8. Jan Lönnberg et al, Java replay for dependence-based debugging, *PADTAD '11* (2011) 15-25.
9. W Furnas, Generalised sheye views, In *Proc ACM SIGCHI 86 Conference on Human Factors in Computing Systems* (1986) 16-23.