

GUI Based Environment to Support Writing and Debugging Rules for a Program Visualization Tool

Daiki TEZUKA ^{a*}, Satoru KOGURE ^b, Yasuhiro NOGUCHI^b, Koichi YAMASHITA^c
Tatsuhiko KONISHI ^b & Yukihiro ITOH^d

^aGraduate School of Integrated Science and Technology, Shizuoka University, Japan

^bFaculty of Informatics, Shizuoka University, Japan

^cFaculty of Business Administration, Tokoha University, Japan

^dShizuoka University, Japan

*gs16027@s.inf.shizuoka.ac.jp

Abstract: TEDViT is a program visualization tool that allows teachers to apply an intention of description (IOD). Using this tool, teachers write a rule set to apply IOD. However, it takes a relatively long time to write the rule set. In this paper, we measure the amount of time required to write rule sets by conventional interface. We determine the reasons why it takes long time and suggest solutions. Thus, we constructed a supporting system that has features corresponding to these solutions. Our experimental results show that the system reduces total writing and debugging time by 41%.

Keywords: Program visualization tool, rule editor, GUI


1. Introduction

It is said that visualizing the behavior of programs is an effective way for novice programmers to understand algorithms. There are well-known program visualization tools such as Jeliot 3 (Moreno et al., 2004), ANIMAL (Rößling et al., 2002), and TEDViT (Yamashita et al., 2015). In particular, TEDViT has distinctive features that allow teachers to apply intentions of description (IODs). The teacher writes T-Rule for applying IOD. A T-Rule can be classified roughly into three types: a creating rule, updating object and deleting object. Figure 1 shows an example of a T-Rule. For example, the teacher can connect two distinct objects, supply the objects with descriptions using balloon objects, and change the colors of the objects. In other words, the teacher can lay out the appearance of objects in a flexible manner.

However, it is difficult for teachers to write T-Rule for TEDViT. Thus, we have developed a support system for writing and debugging a T-Rule set. This is aimed at reducing the burden on the teacher. Figure 2 shows the relationship between the system and TEDViT. In order to design the system, we observed problems that occurred when writing T-Rule sets, analyzed those problems, and proposed improvements to the system. We conducted an experiment and confirmed that our system can shorten the time required for writing T-Rule sets by approximately 41%.

An example of T-Rule (A Create rule)

rule, state==6, **create**, LOOKOBJ, rectangle, look, x1, y1, black, **white**, black

→ look  *Appear*

Create an object on (x1, y1), when the statement 6 is executed. The name of the object is "LOOKOBJ", form is rectangle, a matched variable's name is "look", and linecolor/backcolor/fontcolor are black/white/black respectively, on (x1, y1) when a statement number equals 6.

Figure 1. An example of a T-Rule.

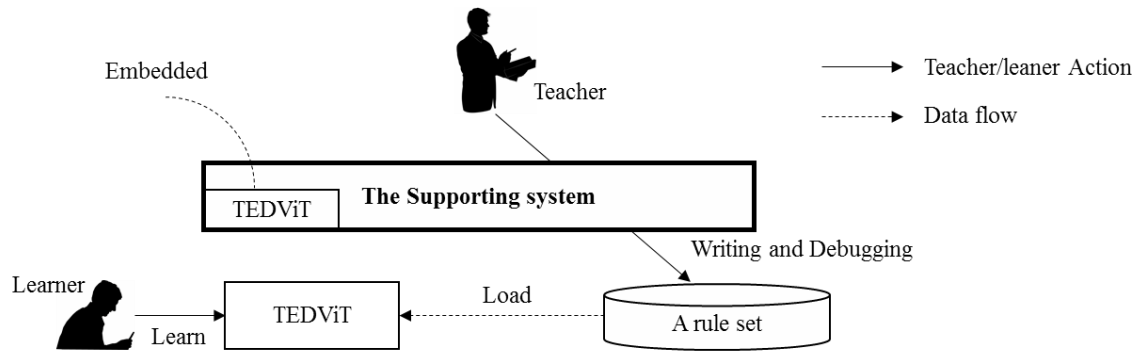


Figure 2. Relationship between the supporting system, TEDViT and a T-Rule set.

2. System Design

We examined problems that occur when writing T-Rule sets. First, we wrote nine T-Rule sets. These rules corresponded to nine algorithms such as binary search and quick sort. The total time for writing these rules was 18 h 47 min. Next, we discussed the problems that occur when writing T-Rules. We found nine problems and classified them into the three types: “Excess/shortage of information content”, “Being unable to operate objects in GUI” and “Other problems”. Finally, we drew up solutions to solve the problems and developed nine features corresponding to each problem. Due to limitations of space, we show only three features.

Feature.1. Teachers often mistype the values of an object’s position because these values must be entered manually. We provide a feature in which the teacher can specify an object’s position by using mouse click. Then, the frequency of mistakes is reduced. When a teacher clicks Area 4 in Figure 3, the X/Y coordinates of a mouse click are converted to a grid format (e.g., “x1,” “y1”). This value is applied to a new T-Rule. Also, the teacher can change an object’s position by using drag-and-drop.

Feature.2. Suppose that the teacher creates T-Rules in order to change a state of an object (e.g., color) at every branches under a selective statement. In such a situation, the teacher must create very similar T-Rules corresponding to each branch. The creating task can be supported by generating such T-Rules by our system. We provide a feature in which the support system specifies a statement number that corresponds to a branch destination, and generates T-Rules whose conditional property value is a number. For example, when a “5” statement jumps to 6 or 10, and a teacher enters “5” in a text area, the system adds T-Rules whose conditional property values are 6 or 10.

Feature.3. It is difficult to notice T-Rules that contain errors until the teacher debugs the statements, even if those errors are simple. We provide the feature in which the support system highlights T-Rules containing syntax errors. Then, the teacher will easily notice the errors.

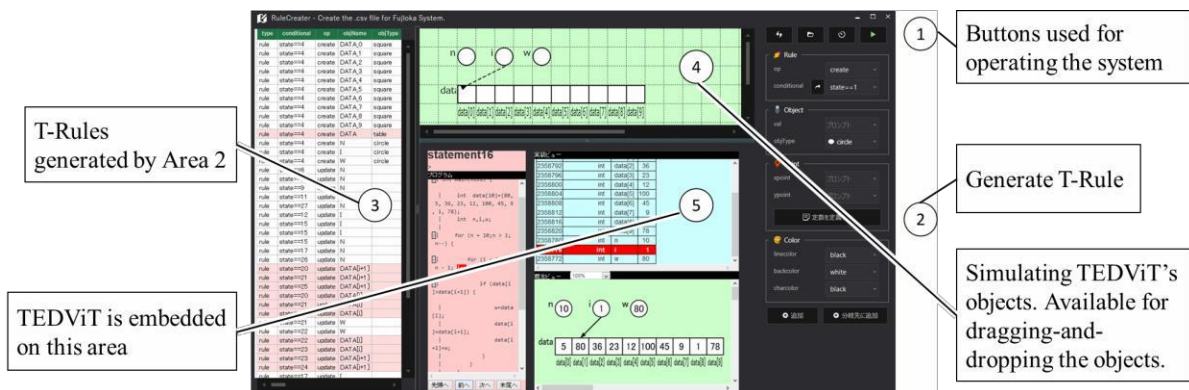


Figure 3. A screenshot of our proposed environment.

We developed the support system using C# language and .NET Framework, which is a software platform developed by Microsoft. A teacher can complete writing and debugging T-Rule sets with only this system. Figure 3 shows a screenshot of the system.

3. Experimental Evaluation

We gathered 8 subjects who have been experienced a teaching assistant of programming class. First, we instructed the subject to write a T-Rule set that behaves like a model T-Rule that we prepared. The model T-Rule is written such that it contains all types of objects that the teacher can use. We explained how to use the support system to each subject. These instructions took approximately 1 h. Then, the subjects start writing and debugging T-Rule sets. Themes for writing and debugging the T-Rule set are “Determining a maximum value” (Theme A) and “Linear search” (Theme B). Each subject wrote and debugged T-Rule sets for the different themes. 4 subjects wrote and debugged T-Rule sets for theme A with our system and for theme B without the system. The other 4 subjects are vice versa.

Table 1 shows all subjects’ total time of writing and debugging T-Rule sets. The value in “Ratio” column shows that our supporting system reduces by 41% of total time. However, the number of subjects was limited.

Table 1: Total time of writing and debugging a T-Rule set.

	Theme A	Theme B	Total	Ratio
With system	1 h 55 min	2 h 54 min	4 h 49 min	<i>0.59 : 1</i>
Without System	4 h 07 min	4 h 02 min	8 h 09 min	

4. Conclusion

In this paper, we developed a system to reduce the time spent on writing and debugging a T-Rule set. The results of our experiments suggest that the system can reduce that time. Also, we used a questionnaire to ask for the subjects’ impressions after using the system. According to subjects’ opinions in the questionnaire, the features of the system is effective. Going forward, we will improve the system based on the results of a questionnaire and teachers’ opinions.

Acknowledgements

This study was supported by Japanese Grant-in-Aid for Scientific Research (B) 24300282.

References

- Moreno, N., Sutinen, M. E.. (2004). Visualizing programs with Jeliot 3. AVI 04: Proceedings of the Working Conference on Advanced Visual Interfaces, 373–376.
- Rößling, G., Freisleben, B. (2002). ANIMAL: A system for supporting multiple roles in algorithm animation. Journal of Visual Languages & Computing, 341–354.
- K. Yamashita, R. Fujioka, S. Kogure, Y. Noguchi, T. Konishi, Y. Itoh. (2015). Educational Practice of Algorithm using Learning Support System with Visualization of Program Behavior. Proceedings of the 23rd International Conference on Computers in Education, 632-640.