Shih, J. L. et al. (Eds.) (2019). Proceedings of the 27th International Conference on Computers in Education. Taiwan: Asia-Pacific Society for Computers in Education

Extending Program Visualization System Based on Teacher's Intent of Instruction to Support Learning Dynamic Data Structures

Koichi YAMASHITA[a*], Yuta HIRAMATSU[b], Satoru KOGURE[b], Yasuhiro NOGUCHI[b], Tatsuhiro KONISHI[b] & Yukihiro ITOH[c]
[a]*Faculty of Business Administration, Tokoha University, Japan*
[b]*Faculty of Informatics, Shizuoka University, Japan*
[c]*Shizuoka University, Japan*
*yamasita@hm.tokoha-u.ac.jp

Abstract: In this paper, we describe an extension of TEDViT, the program visualization system that can reflect teachers' intent of instruction to support learning dynamic data structures. We extend TEDViT to enable the definition of drawing rules using values of target program variables and pointers corresponding to the execution status when the rule is fired. The evaluation results reveal that our extension could significantly reduce the amount of drawing rules and the time required for defining program visualizations. It suggests that our approach would support program visualizations with dynamic data structures to a certain degree.

Keywords: Programming education, program visualization system, program visualization design, dynamic data structures

1. Introduction

To date, several program visualization (PV) systems have been developed to support novices who are learning programming (Pears et al., 2007). We have developed a PV system called Teacher's Explaining Design Visualization Tool (TEDViT) that allows teachers to provide not only a target program, but also its visualization policy (Kogure et al., 2014). The teacher can define, by providing the drawing rules to TEDViT, which variable is visualized with which drawing objects and when they are visualized in the process of program execution. However, TEDViT becomes complex when it comes to visualizing program behaviors with dynamic data structures (DDS) because defining using drawing rules is costly.

In this paper, we describe an approach to extend TEDViT so that it can visualize the program behaviors with DDS. The reasons for the high cost of defining drawing rules on DDS in TEDViT are difficulties in expressing a certain location of main memory for drawing rules and assigning a certain drawing position of objects. To address these issues, we extend the drawing rule representations by enabling the possibility of expressing certain data on memory in a more abstract way. We also increase the abstraction level for position expression of drawing objects, thereby enabling the possibility of assigning a relative position to other drawing objects.

This paper proves that our extension could enable the visualization of program behaviors processing linked list, which is one of the typical DDS, at relatively low cost. We evaluate how much cost is reduced when defining a drawing ruleset in our extension. The evaluation results show that our extension could significantly reduce costs when defining a drawing ruleset in extended TEDViT.

2. Visualizing Behaviors of Programs with Dynamic Data Structures

TEDViT allows teachers to define the policy for drawing a status of the target domain world based on their intents of instruction. Teachers can create or edit a configuration file independently from the target program file, which defines a visualization policy comprises a set of drawing rules, each of which is a CSV entry as in Figure 1. The drawing rule shown in Figure 1 means that when the statement with ID

"10" in the target program is executed, TEDViT draws a circle object and assigns the object ID "OBJ1" to it. The corresponding variable is indicated as "*i*," and hence the value of the variable *i* is drawn inside OBJ1. OBJ1 is drawn at position (x1, y1) with a line, background, and inner character colors of black, white, and black, respectively, according to the indicated values in the rule.

```
state==10, create, OBJ1, circle, i, x1, y1, black, white, black
```

Figure 1. Example of the drawing rule in TEDViT.

When defining PV with DDS, teachers often demand the dynamic location of drawing objects depending on the program behaviors. For example, in a linked list node insertion, the target node to be inserted is naturally drawn near the linked list, at the position where the target is inserted into. Teachers may also demand to highlight some drawing objects according to specific timing, to provide natural-language explanations by accompanying some objects with balloon messages, etc. However, the original TEDViT required that the drawing objects and their positions be statically indicated by object ID and absolute coordinates in pixel or grid measure. Hence, we extend TEDViT to be capable of indicating drawing objects by using values of target program variables corresponding the execution status when the rule is fired, and of indicating coordinates relative to any other drawing objects in drawing rules. In drawing rule description, teachers can describe the following functions to indicate target objects and their positions that are origins of relative coordinates;

- xname(*ObjectID*), yname(*ObjectID*): finds x-/y-coordinate of the drawing object with the ID *ObjectID* given by the argument.
- xvari(*VariableName*), yvari(*VariableName*): finds x-/y-coordinate of the drawing object corresponding to the variable with the name *VariableName* given by the argument.
- xaddr(*PointerExpression*), yaddr(*PointerExpression*): finds x-/y-coordinate of the drawing object corresponding to the memory location referred by the pointer *PointerExpression* given by the argument. The value of *PointerExpression* depends on the execution status when the rule is fired.
- searchObjectbyValue(*PointerExpression*): finds drawing object corresponding to the memory location referred by the pointer *PointerExpression* given by argument.
- searchObjectbyName(*VariableName*): finds drawing object corresponding to the memory location that stores the value of variable with the name *VariableName* given by argument.

In rule definition, teachers sometimes need to describe variable names for indicating memory locations corresponding to drawing objects. We also extend TEDViT to be capable of describing the following function:

- searchVariable(*PointerExpression*): finds variable name corresponding to the memory location referred by the pointer *PointerExpression* given by the argument.

3. Experiment

Since PVs generated by the extended TEDViT basically follow the ones generated by original TEDViT, it is expected that the extended TEDViT would provide the same level of learning effects as the original TEDViT. Therefore, we conducted an experiment to evaluate the cost to define drawing rules for PV with DDS. The number of subjects were 8: two teachers who have experience of teaching programming, two students who have experience as programming teaching assistants (TAs), and four students who have the same level of programming experience as programming TAs.

At the beginning of the experiment, we explained an outline of the experiment, the defining of PVs, and the use of original and extended TEDViT to the subjects, for 30 min. Then, the subjects defined PVs of a program consisting of about 40 statements. They defined PVs in two-time intervals, first with the original TEDViT in 60 min, and then with the extended TEDViT in 60 min. To reduce the order effects, we divided the subjects into groups A and B, each of which included one teacher, one TA, and two students, and swapped the order in which they used the original and the extended TEDViT. The subjects in group A used the original TEDViT first, while those in group B used the extended TEDViT first. For each of the two rule definitions, we measured the minutes taken to define entire drawing rules and the number of defined rules for every subject.

Table 1 shows the results of the experiment. The value in parentheses that is appended below the rule definition time with original TEDViT indicates the time estimated to complete the entire rule definition. Given that the subjects had to define the drawing rules within a time limit of 60 min, no subject except #8 could complete their rule definition within that time. We estimated the ruleset completion times using the ratio of the number of the total rules we expected to be answered to the number of drawing rules defined in 60 min.

Table 1

The Defining Time and the Number of Drawing Rules

Subject		#1	#2	#3	#4	#5	#6	#7	#8
Group		A	A	A	A	B	B	B	B
N of rules	w/ Original	46	40	58	39	42	18	18	70
	w/ Extended	8	8	10	8	8	8	8	8
Time (min.)	w/ Original	60*	60*	60*	60*	60*	60*	60*	55
	(Estimated)	(88)	(102)	(70)	(104)	(97)	(226)	(226)	
	w/ Extended	17	29	26	21	30	51	59	23

("*" represents the time for subjects who did not finish the rule definitions.)

We can see that the numbers of drawing rules to define PV and the rule definition intervals are significantly reduced by using the extended TEDViT. Note also that the average number of rules in the extended TEDViT is 24.8% of those in the original version. The same tendency can be seen in group A, 18.8%, and group B, 29.8%. Based on the estimated times, the subjects could have completed their rule definition with the extended TEDViT in 28.3% of the interval required with the original, on average. The same tendency also can be seen in group A, 26.2%, and group B, 30.0%. These results suggest that the cost to define PVs with DDS could be significantly reduced by our extension.

4. Conclusion

In this paper, we describe an extension of a PV system called TEDViT to enable the visualization of program behaviors with dynamic data structures. We extend TEDViT to enable the possibility of specifying the target and its position through values of target program variables corresponding the execution status when the rule is fired. It means to enhance the abstraction level of rule descriptions, such as specifying a drawing object referred by a certain pointer, specifying a drawing position like "in the vicinity of a certain object," etc. The evaluation results revealed that the extended TEDViT could significantly reduce the amount of drawing rules and the time to define PVs likewise for both groups. This suggests that the extended TEDViT could enable to visualize the program behaviors processing linked list, which is a typical DDS, at relatively low cost. Therefore, it can be concluded that our extension would help to define TEDViT PV with DDS to a certain degree.

Acknowledgements

This study was supported by JSPS KAKENHI Grant Numbers JP16K01084, JP18K11567, and JP19K12259.

References

Kogure, S., Fujioka, R., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2014). Code reading environment according to visualizing both variable's memory image and target world's status. Proceeding of the 22nd International Conference on Computers in Education, 343-348.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A Survey of Literature on the Teaching of Introductory Programming. *ACM SIGCSE Bulletin, 39*(4), 204-223.