Duplicated Code Slicing Technique for System Optimization

Seunghyung Lee, Sungho Sim

Abstract: ***These days, the systems have been bigger upon integrations with multiple functions of hardware and software. To optimize these bigger systems, slicing technique is required to extract the duplicated codes. In this study, system dependent graph was used for slicing of duplicated codes. System dependent graph is generated upon analysis of extracted control relationship from system codes and data dependence. Duplicated control and data relations are extracted upon analysis of generated system dependent graph. Using control and data relations, which is a suggestive slicing technique, duplicated codes can be generated. Slicing technique using system dependent graph can be applied to the extraction of duplicated cross cutting modules in all programming methods regardless of the environment of structure/object-oriented program. By the suggestive method, code blocks duplicated. In the system can be sliced and system code optimization can be contributed by eliminating unnecessary codes from slicing.***

Keywords :Control Dependence Relation, Duplicated Code Slicing, System Optimization, Data Dependence Relation, Source Code.

I. INTRODUCTION

Refactoring is a process to change the code structures to enhance the internal structure and reusability without changing program functions and behaviors. In the existing object-oriented method, it is efficient to make modules of core codes but has difficulty in cross-cutting modulization[1]–[3]. Cross-cutting modulization can be separated conceptually in the design stage, while it is complicated in the realization stage since cross-cutting modules are coexisted in the system[4,5].

To optimize the realized system, cross-cutting modules should be extracted and reorganized. This method violates the object-oriented principles, which should use the slicing technique of duplicated source codes[6,7]. In case of using slicing technique for system optimization, duplicated codes in the system can be defined with the cross-cutting area and eliminated easily. Eliminated cross-cutting module can optimize the system upon application of aspect-oriented programing. Existing system optimization techniques were listed considering duplicated source codes within the realized system details. Also, parts of object-oriented system were defined as aspect-oriented programing details to separate specific elements and applied to system optimization. This method is not enough as the objective system optimization method on the object-oriented programs designed and realized.

To optimize the system, the technique to extract duplicated cross cutting modules is essential and the studies are required on the source code slicing technique to extract cross cutting modules. In the suggestive article, system dependence uses the relations of control dependence and data dependence for the slicing of duplicated cross-cutting modules[8]–[10]. Objective approaching methods by stages were summarized for source code slicing using program dependences[11].

II. SYSTEM DEPENDENCE

Program dependence which demonstrates the dependence relation on a procedure is required to analyze the dependence of overall system[12]. Dependence has two aspects including control dependence to show how a process is continued and data dependence to show which variable influences to determine its values.

Control dependence which defines the consecutive processes during the program execution can identify the basic information for code scheduling upon analysis of dependence within the program[13]. It is mainly used in compiler and essential for source code slicing. If each node is connected to the dependent node during its execution process and execution of A code is determined by B code, A is considered as control dependence under B.

Data dependence shows the relations among variables to be influenced by control flow and information can be identified on the parameter process according to the method calling using this relation. Data flow that used in each execution sentence is expressed as the link to the execution order made in the control flow relation[14].

Dependence graph without calling relation among functions is called as program dependence graph, and that including calling functions of the program with the subjects of total sources is called as system dependence graph[15,16]. Upon analysis of dependence relations on each procedure and combination of these, system dependence graph can be generated. System dependence graph is the one that combines with control dependence and data dependence relations. If control dependence and data dependence relations are analyzed, information on the execution time action can be inferred without execution of the system and it can be utilized in the optimization of compiling process. In addition, it can be

Revised Manuscript Received on July 22, 2019.

Sungho Sim, College of General Education, Semyung University, Jecheon-si, Korea. Email:shshim@semyung.ac.kr

Seunghyung Lee, Dept. of Computer Engineering, Kyung Hee University, Youngin-si, Korea. Email: shlee7@khu.ac.kr

applied in the system slicing.

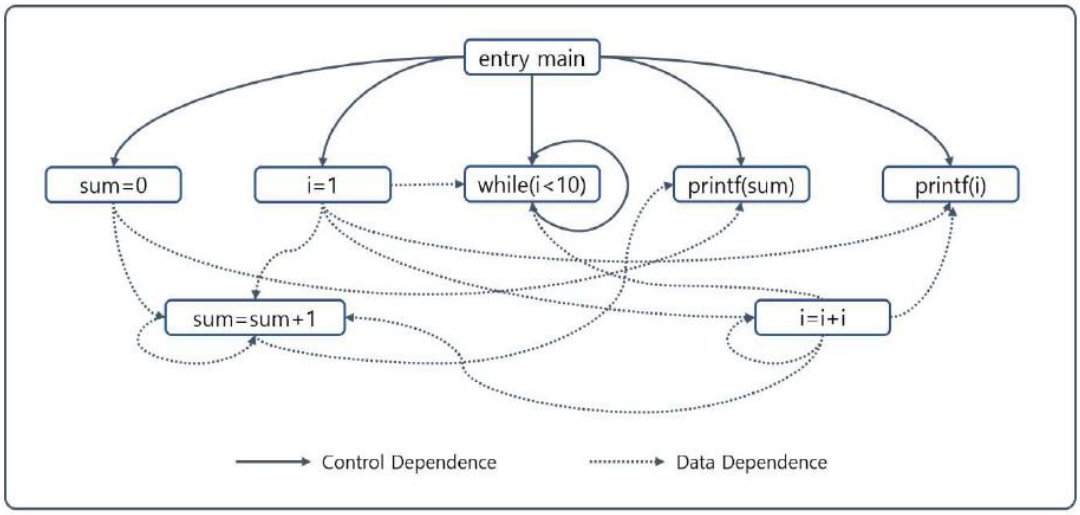

Fig. 1. System dependence graph including control/data dependence relations.

III. PROCESS FOR SYSTEM SLICING

For optimization of complicated system, separations of duplicated cross-cutting modules are essential. System optimization using cross-cutting concept means to optimize the duplicated codes which are the subjects of cross-cutting in the final realized system upon their extraction automatically. To separate cross-cutting module, system slicing technique is applied. Slicing process is designed as Fig 2 to extract cross-cutting modules using system dependence graph.

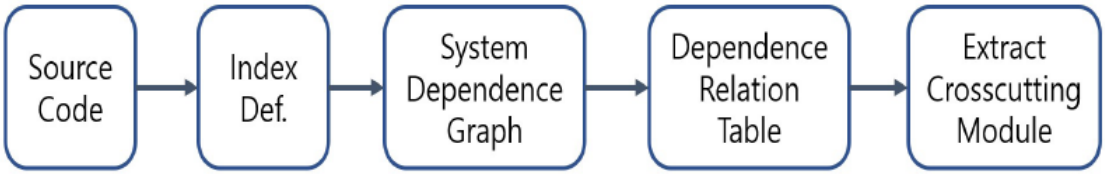

Fig. 2. Process for System Slicing.

In the first step, indexes are defined to generate system dependence graph from total system codes. Individual marks for indexes are allocated for each line of system code which will be demonstrated with the graph of program dependence relations. Lines which can be marked with the same pattern in the dependence graph are allocated with the same marks.

In the second step, system dependence graph is prepared. Control and data dependence relations are described as the directional graph on the system codes by objects. Individual marks are allocated for each line to identify the types of dependences (control and data).

In the third step, table of dependence relations is prepared. Upon classification of dependence relations per each object based on directional graph formula, lines with dependence relation are expressed as Node (N) and relations between nodes are expressed as Edge (E).

$$e_1 = (n_1, n_2), e_2 = (n_3, n_4), e_1, e_2 \in E \quad (1)$$

In the fourth step, duplicated cross-cutting modules are extracted by comparisons of order expressed as Edge. Edges of the objects A and B are defined as $A_e = \{e_1, e_2, \cdots, e_n\}$ and $B_e = \{e_1, e_2, \cdots, e_n\}$. Duplicated source codes are extracted upon combination of connected sources with indexes of each node $N_d = {}^{\forall}n$: $n \in E_d$ which is duplicated edge $\{E_d \mid E_d = Ae, E_d = B_e\}$.

IV. EXTRACTION OF CROSS-CUTTING MODULE BY SYSTEM SLICING

It shows the suggestive process of cross-cutting extraction for system optimization. System example codes are prepared as Fig 3 to apply system slicing. Example code is a partial code of Bison which transforms the generator into JAVA language with GNU. This consists of two classes. To perform system optimization using two classes, duplicated cross-cutting area is extracted by system slicing technique.

```
Public class Bison_OutputA {
  public void token_actions( ) {
    int l; int j; int k;
    actrow = NEW2(ntokens, 2);
    k = action_row(0);
    System.out.println("nstatic const short yydefact[ ] = " + new String(k));
      save_row(0);
      j = 10;
      for(i = 1; i < nstates; i++) {
        System.out.print(",");
        if (j >= 10) {System.out.print("₩n"); j=1;}
        else {j++;}
        k = action_row(i);
        System.out.println(new String(k));
        save_row(i);
      }
    System.out.println("₩n};₩n");
  }
}
```

```
Public class Bison_OutputB {
  public void output_stos( ) {
    int l; int j;
      System.out.println("nstatic const short yydefact[ ] = {0");
      j = 10;
      for(i = 1; i < nstates; i++) {
        System.out.print(",");
        if (j >= 10) {System.out.print("₩n"); j=1;}
        else {j++;}
        System.out.println(new String(accessing_symbol[i]));
      }
    System.out.println("₩n};₩n");
  }
}
```

Fig. 3. Codes to be used system slicing.

Indexes applied with sentence patterns from system code are defined for system slicing. Source code with the same sentence pattern uses the same index. Sentence patterns of two methods which are the subjects of system optimization are defined and individual indexes for all the defined patterns are allocated as Fig 4. Individual indexes are used to map the node in the program dependent graph with source code. Space and some control marks ({}...} are excluded.

Index	Source Code	Index	Source Code
A	int l;	K	if (j >= 10)
B	int j;	L	System.out.print("₩n");
C	int k;	M	j=1;
D	actrow = NEW2(ntokens, 2);	N	j++;
E	k = action_row(0);	O	k = action_row(i);
F	System.out.println("nstatic const short yydefact[ ] ...	P	System.out.println(new String(k));
G	save_row(0);	Q	save_row(i);
H	j = 10;	R	System.out.println("₩n};₩n");
I	for(i = 1; i < nstates; i++) {	S	System.out.println("nstatic const short yydefact[ ] ...
J	System.out.print(",");	T	System.out.println(new String(accessing_symbol[i]));

Fig. 4. Definition of indexes on source codes.

Program dependence relations are analyzed by connecting system codes with index values defined by sentence pattern analysis. With respect to system dependence relations, data and control dependence relations are analyzed based on the direction of system code lines allocated with indexes. System dependence relations of system code lines which are the analysis subjects are generated through this. System dependence graph is organized with control and data dependence relations. Fig 5 shows the table for data and control dependence relations upon analysis of two classes.

Bison_OutputA

Control Dependence Relation	Data Dependence Relation
(I, J)	(E, F)
(I, K)	(H, K)
(K, L)	(M, K)
(K, M)	(N, K)
(K, N)	(I, O)
(I, P)	(O, P)
(I, Q)	(I, Q)
(I, O)	(N, N)

Bison_OutputB

Control Dependence Relation	Data Dependence Relation
(I, J)	(H, K)
(I, K)	(M, K)
(K, L)	(N, K)
(K, M)	(I, T)
(K, N)	(N, N)
(I, T)	

Fig 5. Control and data relations extracted per each class.

Program dependent graph is prepared as Fig 6 including control and data dependence relations of Bison_OutputA/Bison_OutputB. Upon identifying the relations of each node in the prepared program dependent graph, node (code) and edge (control and data) are described as Fig 6 by understanding the control and data dependence relations. Solid lines mean control dependence relations while dotted lines do data dependence relations.

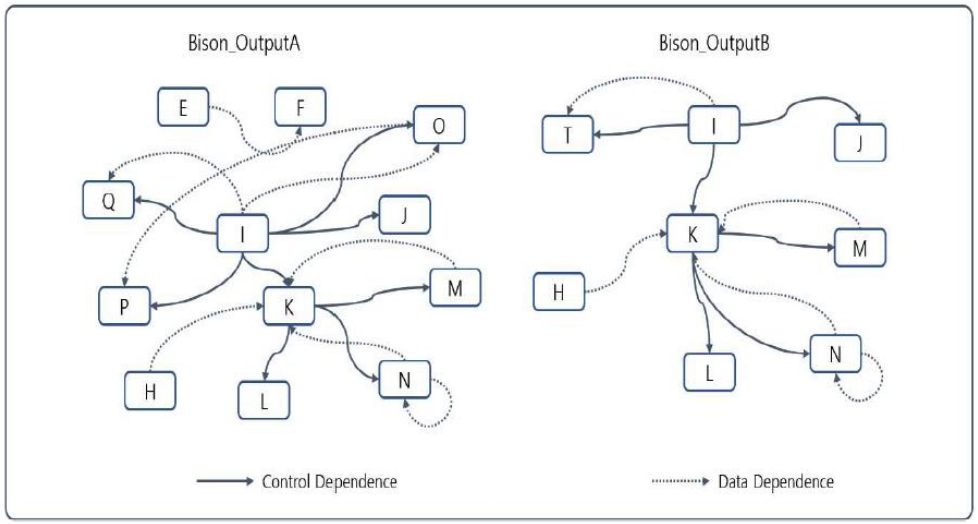

Fig. 6. System dependence graph on each class.

Duplicated edges are important information in slicing to extract duplicated cross-cutting modules. Duplicated edges are extracted upon analysis of dependence relations for extracted classes.

Duplicated edges are important information in slicing to extract duplicated cross-cutting modules. Duplicated edges are extracted upon analysis of dependence relations for extracted classes.

Duplicated edges, $\{E_d \mid E_d \in A_{ec},\ E_d \in B_{ec}\}$, are extracted from control dependence edge of Bison_OutputA class,

$A_{ec} = \{(I,J),(I,K),(K,L),(K,M),(K,N),(I,P),(I,Q),(I,O)\}$

and those of Bison_OutputB class, $B_{ec} = \{(I,J),(I,K),(K,L),(K,M),(K,N),(I,T$.

Duplicated edges are extracted with the same method in data dependence edge. Extracted duplicated edges are as Fig 7.

Duplicated Relation

Control Dependence Relation	Data Dependence Relation
(I, J) (I, K) (K, L) (K, M) (K, N)	(H, K) (M, K) (N, K) (N, N)

Fig. 7. Extractions of duplicated control and data dependence relations.

After extraction of nodes, $\forall n : n \in E_d$ from duplicated edges, sliced cross-cutting modules are generated as Fig 8 upon mapping with source codes of indexes, {H, I, J, K, L, M,

```
j = 10;
for(i = 1; i < nstates; i++) {
System.out.print(",");
if (j >= 10)
System.out.print("₩n");
j=1;
j++;
```

Fig. 8. Sliced modules by analysis of system dependence relations.

V. CONCLUSION

In the structure/object-oriented program whose developments are completed, it has significant difficulty in maintenance and reusability in case of new data introduction or update of functions. In addition, it is not possible to reorganize the specific elements only within the capsuled objects after slicing in the object-oriented programing. Hence, lots of difficulties are imposed in the treatments of duplicated codes which are common elements in each object. There is no objective approaching method to solve this problem.

In this article, slicing method for duplicated cross-cutting modules was suggested which could apply in both structure and object-oriented program environments. To extract the reusable modules, control and data dependence relations were used. Individual indexes for each system code were defined. Based on the defined indexes, system dependent graph was generated upon investigating control and data dependence relations. Compared to system dependent graph, table

for control and data dependence relations was generated. By comparison of orders in the generated dependence relation table, duplicated elements could be extracted, and duplicated modules could be generated by extracted relations. With the suggestive method in this article, slicing problem of the capsuled objects could be solved without limitation of structure/object-oriented program environment, and this could contribute to optimize the complicated system.

APPENDIX

It is optional. Appendixes, if needed, appear before the acknowledgment.

ACKNOWLEDGMENT

It is optional. The preferred spelling of the word "acknowledgment" in American English is without an "e" after the "g." Use the singular heading even if you have many acknowledgments. Avoid expressions such as "One of us (S.B.A.) would like to thank ... ." Instead, write "F. A. Author thanks " *Sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page.*

REFERENCES

1. L. Aversano, M. Cerulo, MD. Penta, "How clones are maintained: An empirical study," In Proceedings of the 11th European Conference on Software Maintenance and Reenfineering, 2007, pp. 81-90, https://doi.org/10.1109/CSMR.2007.26
2. Misael Mongiovì, Giuseppe Pappalardo. Emiliano Tramontanab, "Specifying and identifying widely used crosscutting concerns," Knowledge-Based Systems, Vol. 126, 2017, pp. 20-32. https://doi.org/10.1016/j.knosys.2017.04.003
3. M. Marin, L. Moonen, A. Van Deursen, "An integrated crosscutting concern migration strategy and its application to JHotDraw," Seventh IEEE International Working Conference on Source Code Analysis and Manipulation, 2007, pp. 101-110. https://doi.org/10.1109/SCAM.2007.25
4. D. Binkley, S. Danicic, T. Gyimothy, M. Harman, A. Kiss, B. Korel, "A formalization of the relationship between forms of program slicing," Science of Computer Programming, Vol. 62, No. 3, 2006, pp. 228-252. https://doi.org/10.1016/j.scico.2006.04.007
5. Shailendra Narayan, Leena, "Study of current program slicing techniques," 5th International Conference-Confluence The Next Generation Information Technology Summit, 2014, pp. 810-814. https://doi.org/10.1109/ CONFLUENCE.2014.6949332
6. DamianoZanardini, "The semantics of abstract program slicing," In: Proceedings of the IEEE International Working Conference of Source Code Analysis and Manipulation, 2008, pp. 89-98. https://doi.org/10.1109/SCAM.2008.19
7. Alaknanda Chandra, Abhishek Singhal, Abhay Bansal, "A study of program slicing techniques for software development approaches," International Conference on Next Generation Computing Technologies, 2015, pp. 622-627. https://doi.org/10.1109/NGCT.2015.7375196
8. Durga Prasad MadhusmitaSahu, Rajeev Kumar, Rajib Mall, "Dynamic Slicing of Aspect-Oriented Programs," In Proc of Informatica, Vol. 32, No. 3, 2008, pp. 261-274.
9. D.W. Binkley, M. Harman, "A surver of empirical results on program slicing," Advance in Computers, Vol. 65, 2004, pp. 105-178.
10. H. Subramaniam, H. Zulzalil, MA. Jabar, S. Hassan, "Feasibility Study of Aspect Mining at Requirement Level," Indian Journal of Science and Technology, Vol. 7, No. 5, 2016, pp. 17-23.
11. G. Shu, B. Sun, TA. Henderson, A. Podgurski, "JavaPDG: Anew platform for program dependence analysis," In: Software testing, verification and validation (ICST), IEEE sixth international conference on IEEE, 2013, pp. 408-415. https://doi.org/10.1109/ICST.2013.57
12. S. Sukumaran, A. Sreenivas, R. Metta, "The dependence condition graph: Precise conditions for dependence between program points," Computer Language, Systems & Structure, Vol. 36, No. 1, 2010, pp. 96-121. https://doi.org/10.1016/j.cl.2009.04.001
13. J. Krinke, "Mining control flow graphs for crosscutting concerns," Proceedings of Working Conference on Reverse Engineering, 2006, pp. 334-342. https://doi.org/10.1109/WCRE.2006.37
14. M. Mongiov`I, G. Giannone, A. Fornaia, G. Pappalardo, E.Tramontana, "Combining static and dynamic data flow analysis: a hybrid approach for detecting data leaks in java applications," The Journal of the Korea Xontents Association, 2015, pp. 1573-1579. https://doi.org/10.1145/2695664.2695887
15. MP. Robillard, GC. Murphy, "Concern graphs: finding and describinf concerns using structural program dependencies," Proceedings of International Conference on Sofware Engineering (ICSE), 2002, pp. 406-416. https://doi.org/10.1145/581339.581390
16. R. Halder, A. Cortesi, "Abstract program slicing on dependence condition graphs," Science of Computer Programming, Vol. 78, No. 9, 2013, pp. 1240-1263. https://doi.org/10.1016/j.scico.2012.05.007

AUTHORS PROFILE

SeunghyungLee received the Ph.D. degree from KYUNG HEE University, Seoul, in 2011.

He heen been a Professor in the college of Software Kyung Hee University, Korea, since 2011.

His research interests include Software Component, System Optimization, Web Component Based Software Development and Software Engineering

SunghoSim received the Ph.D. degree from KYUNG HEE University, Seoul, in 2012. He has been a Professor in the College of General Education, Semyung University, Korea, since 2013. His research interests include Web Service, Meta-data, Web component, Web Service QoS, Component based Software development and Software Engineering