An Approach for Displaying the Relations among Main Elements of Object-Oriented Programs

Kayvan Kaseb
University of Kurdistan
Kayvan.kaseb@ieee.org

Mohammad Sayedi
University of Kurdistan
Mohammad.sayedi@gmail.com

Abstract

An appropriate understanding of the source code is one of the necessary steps for resolving errors and improving code and design. Two fundamental aspects in Object-Oriented Programs are program elements including classes and packages, and the relations among them. In this paper, a multi-step approach has been presented and implemented for recovering and displaying main elements of an Object-Oriented Program including classes, packages, and the relations among them. This approach is done in three steps: At first, the set of classes, packages, and the relations among them are obtained automatically from the program source code. In the next step, a code is injected to the program source code to register the information needed during runtime. In the third step, the information obtained from previous steps is displayed. The results of this approach may be used in automatic documentation, teaching programming, easy understanding and evaluating Object-Oriented Programs, reverse engineering methods for detecting program strengths and weaknesses

Key words: Object-Oriented, Program Understanding, Program Evaluation, Program Visualization,

Introduction

Software is usually complex and intangible. It is known that graphical visualization can convey information better and even faster than numerical data. Program visualization is used for reducing source-code complexity and for easing understanding it (Balzer M. and Noack A. 2004), (Stasko, J. 1998). One of the crucial steps for correcting program errors as well as improving program design and code is having a good understanding of the program source-code (Storey M., 2006). Researches have revealed that a good comprehension of the source-code is an essential issue in developing and maintaining software systems (Davis A. M., 1995). Regarding the importance of this matter, numerous researches, techniques, and tools have been developed (Ducasse S. and Lanza M. 2005), (Gracanin D. 2005), (Petre M. and Quincey E. 2006), (Caserta P. and Zendra O. 2011). One of the available tools, Rigi, is a tool that focuses on the complexity of large systems through visual demonstration by graphs (M¨uller H. A 1998). Code Crawler is a context-free environment, used to visualize the

program via Object-Oriented Program graph, providing simple metrics about the program (Lanza M. 2003), (Lanza M. 2004). Creole, is an Eclipse plug-in that displays code using grid, radial, and tree map techniques (Michaud R. 2003). Source Navigator (Source-Navigator-Team 2006) is a tool for analysing and editing source-code and displaying classes and their relationships in form of a tree. Source Viewer 3D (Marcus A. 2003) is yet another tool that displays the source-code in 3D and has been developed following the implementation of SeeSoft (Eick S. G. 1992).

In order to show source-code components and the relations among them, most of the mentioned tools benefit from graph-based visualization. Knowing the fact that reading an object-oriented code is much more complicated than reading a structured code (Dekel U. 2002), the main focus of this work is on object-oriented source-code, however, this work can be also used for structured source-code cases, of course, with some modifications. By examining Object-Oriented Programs, one can understand the two important components of Object-Oriented Programs: program components, and the relations among them. Knowing the specifications of methods, classes, and packages is important in understanding and modifying the program, however, understanding the relations among classes and packages for complex systems is more important (Ducasse S. 2011), due to the fact that the complexity of the system is created based on these relations. In this paper, a three-step method has been presented and implemented in order to recover and demonstrate the main components of an Object-Oriented Program, including classes, packages, and the relations among them. At the first step, the set of classes, packages, and the relations among them will be automatically acquired from the program source-code. In the next step, a code segment will be injected to the program source-code to register the required data; where in the third step the information obtained from the previous steps will be displayed. The results of this method could be used in automatic documentation, learning programming, easy understanding and evaluation of Object-Oriented Programs, as well as reverse engineering in order to identify program weaknesses and strengths.

Object-oriented program source-code analysis

In this level, program source code is read and analyzed based on a special algorithm and the result of this work which is finding classes and packages and the relations among them, is stored in a database. It has been tried to maintain the runtime of this level in an acceptable limit.

Obtaining classes and packages

As the suggested method is suitable for displaying classes and packages, only these two components are extracted. In large programs, displaying other components including methods could be very difficult. We define an Object-Oriented Program as below:

An Object-Oriented Program is a set of packages and classes where each package contains specifications such as name and level. Level of a package is mentioned because packages are nested. For the first level, number one is assigned, and for packages in levels two and more, parent package is considered the one that contains the child package. Each class has specifications such as name and in case of existence, name of the class it has been inherited from, as well as the name of its containing package. According to the definition above, full

name of a package or a class along with the name of its parent can be written from left to right using period (.) sign. This way of writing is commonly used in most Object-Oriented Programming languages.

Finding the relations among classes

References After obtaining program components, it is time to find the relation among them. Fundamental relation among classes is related to a class being used by another one that can occur for the following reasons. If C_i and C_j are two separate classes, then C_i uses C_j if:

- C_i contains an attribute of C_j type.
- A method of C_i has an input argument of C_j type.
- The returned value type of C_i is of C_j type.
- Using C_j static method(s) in C_i
- C_i is inherited from C_j.
-

From now on, we call classes such as C_j that use other classes, as client classes and call classes which are being used, as provider classes.

Finding the relations among packages

Package P_i uses package P_j if at least a class such as C_i exists in P_i that uses C_j class in P_j. From now on, we call packages such as C_j that use other packages and packages which are being used, client and provider packages respectively.

Injecting code to the program

The purpose of this stage is to inject a code to a method so that during method execution time, two types of information will be registered: The time consumed by a method and the number of times a method is called during a single program execution. Having such information provides numerous advantages, e.g. during the process of testing the methods, where methods with long execution time are specified. By enhancing the execution time of methods, the overall program will be improved.

Proposed display pattern

In order to explain the presented solution, packages and classes of a hypothetical program are shown in figure 1. (a) In this figure, directional lines display the relation among classes, from client to provider. The relation among packages is obtained from the relation among classes in table 1.

In order to display classes and packages, a special pattern is proposed in a way that for all display components, client component is drawn at the left while the provider is drawn at the right, both from top to down. A horizontal line among the client and the provider shows the relation among them. As a result, the created display is a tree that is seen 2D, from left to right and top to down.

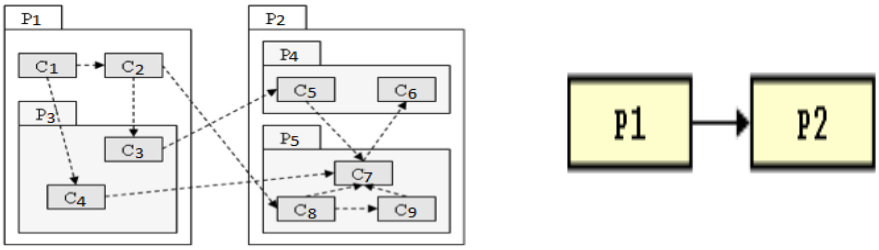

Fig.1. (a) Relations among classes and packages in a hypothetical program; (b) The relation among packages in the first level

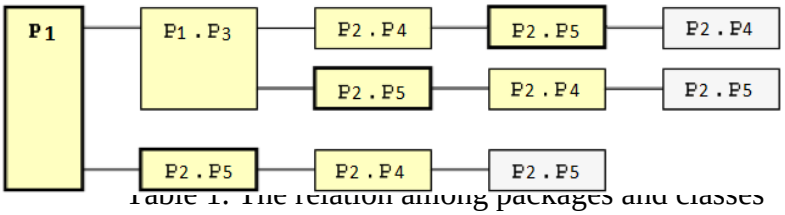

Table 1. The relation among packages and classes

Relation Between Classes	Relation Between Packages	Relation Between Packages in the First Level
C1→C2	P1→P1	P1→P1
C1→C4 C2→C3	P1→P1.P3	P1→P1
C2→C8	P1→P2.P5	P1→P2
C3→C5	P1.P3→P2.P4	P1→P2
C4→C7	P1.P3→P2.P5	P1→P2
C8→C9 C9→C7 C8→C7	P2.P5→P2.P5	P2→P2
C5→C7	P2.P4→P2.P5	P2→P2
C7→C6	P2.P5→P2.P4	P2→P2

Description of the proposed method with regard to the proposed display:

- Display area can be extended both horizontally and vertically. By providing this capability, compatibility of display with larger programs will be achieved.
- The placement of components in this case prevents the joining lines from crossing each other.
- In this display, the order of calls is not important, because in a large display, relations are of more importance than their order.
- Relations are shown from top to down. This means that at the beginning, first level packages are displayed (figure 1. (b)). by choosing the desired package, the user can then see the relation among next level packages that are related to the selected package.
- Sometimes the relations among the components may create a loop. For example, relations P1.P5 and P1.P4 create a loop. In this case, the component that creates the loop could be displayed in gray color. This makes loops detectable in the display and maintains display collectivity.
- It is probable for a component to have a relation with itself, meaning that it may have internal relations. In these cases, the rectangular border can be displayed bolder.
- In order to improve display, the user can determine settings such as size and type of font used to write text, rectangle size, and distance among rectangles. We include these items in the desired display parameters.

Interacting with program display

In this method, the user is able to interact with the display, leading to a better understanding of the display and categorization of display components. In this section, interaction methods are introduced as well as the proposed interaction in order to abbreviate the display.

Hiding components

In order to improve display quality, sometimes it is necessary to abbreviate the display. If B is a provider component for A and we would like to hide it, we can omit the display distance among A and B. For example, if we would like to hide package P1.P3 from figure 2, figure 3. (a) Will be obtained.

Displaying components separately

Another abbreviation method is to display different parts of the image separately. In this case, one or more components could be chosen to be displayed in another page. For example, if we would like to show package P1.P3 separately from figure 2, figure 3. (b) Will be obtained.

Pruning

Although hiding component is a useful solution, however, in hiding methods, display height does not change and only display width decreases. Pruning can be used to abbreviate the display and reducing display height. This is done in a way that for all chosen components, all providers are omitted and only a number showing the number of providers is displayed. For example, if we show package P1.P3 with pruning, figure 4. (a) Will be obtained

Deleting a relation

The increase of program display size is due to the existence of relations among components. Thus one way to abbreviate the display is to delete relations in the display. For example, if we delete the relations among providers of P1, figure 4. (b) Can be obtained.

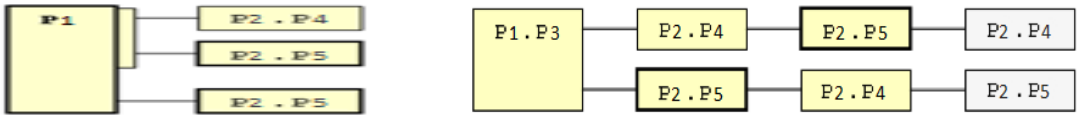

Fig.3. (a) Hiding the distance among P1 and P3 components; (b) Figure 2 after displaying P1.P3 separately

Fig.4. (a) Package P1.P3 with pruning; (b) Figure 2 after deleting the relations among providers of P1 package

Experimental Results

In order to evaluate the proposed method, a tool called PCVis was implemented using C# programming language. As an example, we examined an Object-Oriented Program called CheckMate [18] that has been developed in C# (figure 5). Similar results can be obtained using for other open source Object-Oriented Programs. However, these results could be obtained using other programming languages as well.

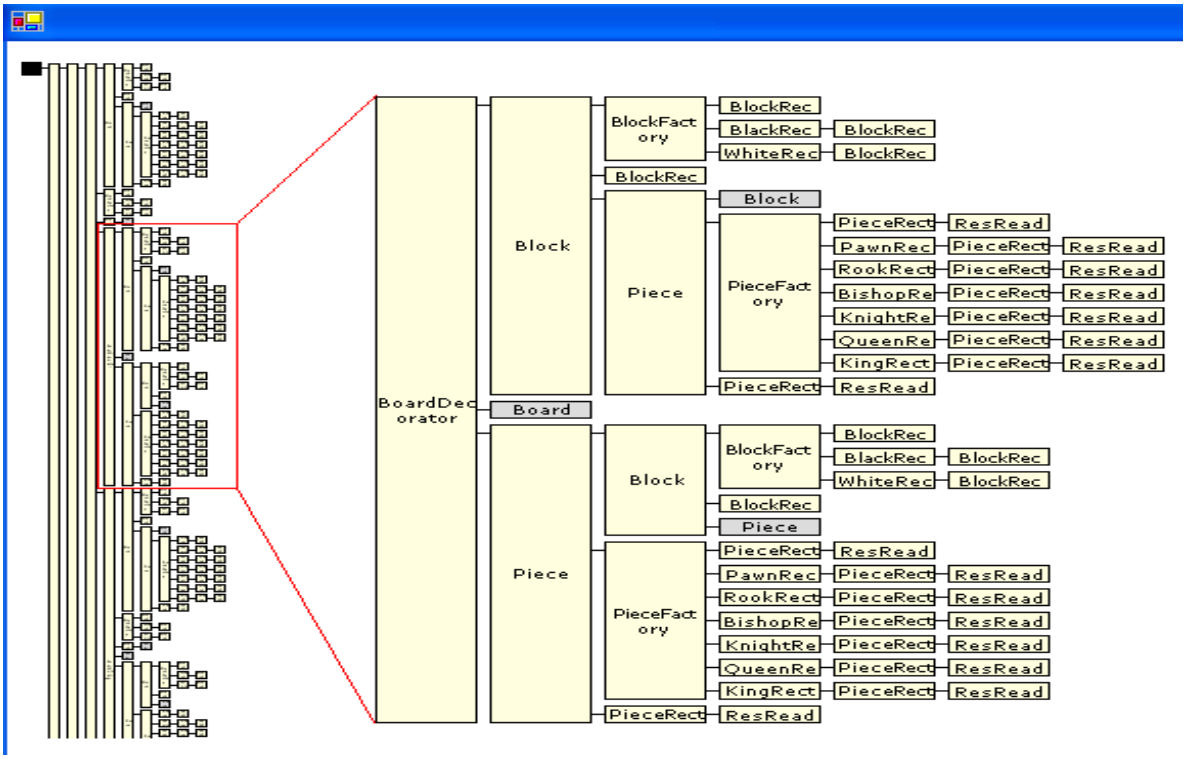

Fig.5. Implementing relations among classes of CheckMate via PCVis tool

Conclusions

In this paper, a multi-step approach was provided for displaying Object-Oriented Programs. This approach may result in recovery of the main components of the program including classes and packages and the relations among them. A graphical representation of the program is then presented based on a special pattern. The main advantages of the implemented approach over similar solutions include generality, being relatively automatic, and expandability, because the only steps that depend on the programming language are the first and second steps. The desired approach could be applied to all Object-Oriented Programs, because it extracts relationships that are common in all Object-Oriented Programming languages. Moreover, because program size is not considered a constraint in using this approach, this approach could be applied to large Object-Oriented Programs as well. Another benefit of this approach is that a vivid and disciplined display of the program is provided and can be used with the rest of program documents.

References

Balzer M. and Noack A. (2004), “Software landscapes: Visualizing the structure of large software systems”, Joint EUROGRAPHICS-IEEE TVGD Symposium on Visualization.

Stasko, J., Domingue, J., Brown, M.H., Price, B.A. (1998), "Software Visualization - Programming as a Multimedia Experience", MIT-Press, Massachusetts.

Storey M., (2006), "Tools and Research Methods in Program Comprehension: Past, Present and Future", Software Quality Control, 14(3), p.187–208.

Davis A. M., (1995), "201 Principles of Software Development", McGraw-Hill.

Ducasse S. and Lanza M. (2005), The Class Blueprint: Visually Supporting the Understanding of Classes. IEEE Transactions on Software Eng., VOL. 31, NO. 1.

Gracanin D., Matkovic K., and Eltoweissy M. (2005), "Software Visualization", Innovation in System and Software Eng, Vol.1, no.2, p.221-230.

Petre M. and Quincey E. (2006), "A Gentle Overview of Software Visualization", PPIG News Letter, Sep., p.1-10.

Caserta P. and Zendra O. (2011), "Visualization of the static Aspects of Software: A Survey", IEEE Transaction on Visualization and Computer Graphics, 17.7, p.913-933.

9M¨uller H. A. and Klashinsky K., Rigi (1998), "A System for Programming-in-the-Large", In Proceedings of the International Conference on Software Engineering, Singapore, IEEE Computer Society Press, p.80-86.

Lanza M. (2003), "Code crawler-Lessons Learned in Building a Software Visualization Tool", Proceedings of the Seventh European Conference on Software Maintenance and Reengineering, p.409-418.

Lanza M. (2004), "Code crawler - Polymeric Views in Action", In Proceedings of the International Conference on Automated Software REngineering, Linz, Austria, IEEE Computer Society Press, p.394-395.

Michaud R., Storey M., and Wu X. (2003), "Plugging-in visualization: Experiences Integrating a Visualization Tool with Eclipse", ACM Symp., on Software Engineering, (Softvis'2003).

13Source-Navigator-Team (2006). Available at: http://sourcenav.sourceforge.net, GNU.

Marcus A., Feng L., and Maletic J. I. (2003), "3D Representations for Software Visualization", In Proceedings of the ACM Symposium on Software Visualization, New York, NY, USA, ACM Press, p.27-36.

Eick S. G., Steffen J. L., and Eric S. E. (1992), Jr. Seesoft—A Tool for Visualizing Line Oriented Software Statistics. IEEE RTransactions on Software Eng., 18(11):957–968.

Dekel U. (2002), "Applications of Concept Lattices to Code Inspection and Review", technical report, Dept. of Computer Science, Technion.

Ducasse S., Pollet D., Suen M., Abdeen H., Alloui I. (2007), "Package Surface Blueprints: Visually Supporting the Understanding of Package Relationships", International Conference on Software Maintenance (ICSM2007).

Available at:
http://www.csharpcorner.com/UploadFile/kaushalgol/ChessProgram11152005054904AM/ChessProgram.aspx.