UNIVERSITI PUTRA MALAYSIA

SLICING OBJECT ORIENTED PROGRAMS FOR MAINTENANCE PURPOSES

HAMED JASEM KHALED AL-FAWAREH

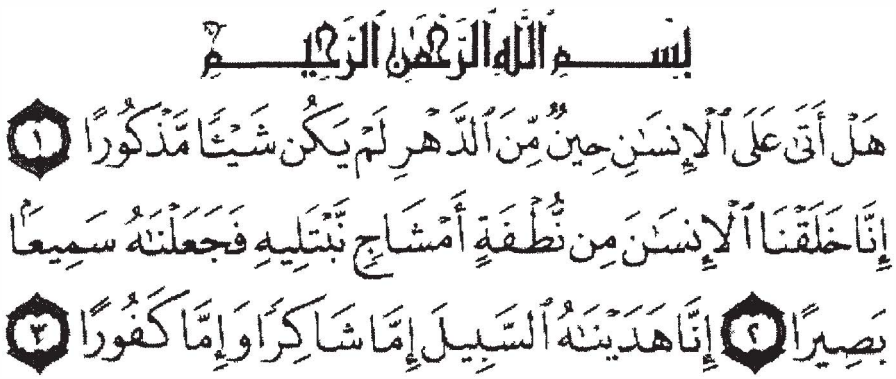
SLICING OBJECT ORIENTED PROGRAMS FOR MAINTENANCE PURPOSES

By

HAMED JASEM KHALED AL-FAWAREH

Thesis Submitted in Fulfilment of the Requirements for the Degree of Doctor of Philosophy in the Faculty of Computer Science and Information Technology
Universiti Putra Malaysia

July 2001


بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

هَلْ أَتَىٰ عَلَى الْإِنسَانِ حِينٌ مِّنَ الدَّهْرِ لَمْ يَكُن شَيْئًا مَّذْكُورًا ﴿١﴾

إِنَّا خَلَقْنَا الْإِنسَانَ مِن نُّطْفَةٍ أَمْشَاجٍ نَّبْتَلِيهِ فَجَعَلْنَاهُ سَمِيعًا

بَصِيرًا ﴿٢﴾ إِنَّا هَدَيْنَاهُ السَّبِيلَ إِمَّا شَاكِرًا وَإِمَّا كَفُورًا ﴿٣﴾

In the name of Allah, the Beneficent, the Merciful.

1 Has there not been over Man a long period of Time when he was nothing-- (not even) mentioned?

2 Verily We created Man from a drop of mingled sperm in order to try him so We gave him (the gifts) of Hearing and Sight

3 We showed him the Way whether he be grateful or ungrateful (rests on his will)

"Holy Quran, Surat Al-Insan Ayah 1-3 "

To

Unconditional (Loves, Dua, Supports, Guidance, and Encouragement)

To My Mother, Eldest Brother Ali, and My Family

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfilment of the requirement for the degree of Doctor of Philosophy

SLICING OBJECT ORIENTED PROGRAMS FOR MAINTENANCE PURPOSES

By

HAMED JASEM KHALED AL-FAWAREH

July 2001

Chairman: Associate Professor Abdul Azim Abd. Ghani, Ph.D.,

Faculty: Computer Science and Information Technology

Object oriented approach is growing very fast in various applications of the computer science. Those applications may contain a lot of entity relationships which, need to be understood by the maintainers. These relationships (involving classes, message, variables, ... etc.) will make maintainers and developers face several problems to understand, make changes, modify, or enhance a huge software system that contains thousands of classes without automatic aids. Therefore several problems arise in the maintenance of object oriented program that is software understanding, complex dependencies, inheritance, polymorphism and dynamic binding.

An approach for formally defining slices for object oriented programs is an important problem in the maintenance phase. This thesis proposes definitions of slices for object oriented programs. Basic relations such as Usage, Affect and Inheritance are defined, and they are extended to a family of dependence relations between entities in object oriented programs and defines slicing techniques based on these relations. Slice

collection methods for specified slice criteria are given. This approach shows how the proposed slicing concepts and rules can be applied within the software maintenance process by giving an illustration through examples written by Java and Delphi programming languages.

The research also develop a prototype of an object oriented system tool (O^2SMt) which represent an automatically extractable and captures an object oriented software system dependencies in order to aid in maintenance phase. The dependencies occurs and explain in this research are *control dependence, statement dependence on a variable, statement dependence on a method, variable dependence on statement, variable dependent on a method, Class-Class dependence through usage, Class-Class dependence through inheritance, Class-Class dependence for causing side effects, method dependence on another method, and method dependence on a variable*. The O^2SMt captures program slicing according to the slicing concepts, rules and definitions to feature out the dependencies with the basic object oriented relations.

This research also, discusses an object oriented dependence graph (O^2DG). The O^2DG categorized according levels. The first category is *class-level* involving a class to another class. The second category is *method-level* involving a method or a statement within a method to another method or variable in a class. The final category is *statement-level* which is basically intra-method involving statements within a given method. Slices, in turn, can also be categorized according to such levels of dependencies they intend to capture.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Doktor Falsafah

SLICING OBJECT ORIENTED PROGRAMS FOR MAINTENANCE PURPOSES

Oleh

HAMED JASEM KHALED AL-FAWAREH

Julai 2001

Pengerusi: **Profesor Madya Abdul Azim Abd. Ghani, Ph.D.,**

Fakulti: **Sains Komputer Dan Teknologi Maklumat**

Pendekatan berorientasi objek berkembang dengan pesat dalam pelbagai aplikasi sains komputer. Aplikasi-aplikasi itu mungkin mengandungi beberapa hubungan entiti yang perlu difahami oleh penyenggara. Hubungan-hubungan ini (termasuk kelas, mesej, pembolehubah, ...dll) akan meyebabkan penyenggara dan pembangun menghadapi beberapa masalah untuk memahami, membuat penukaran, perubahan, atau penambahan terhadap sistem perisian yang besar yang mengandungi ribuan kelas tanpa bantuan automatik. Oleh kerana itu beberapa masalah timbul dalam penyenggaraan atur cara berorientasi objek iaitu kefahama, kebergantungan yang kompleks, pewarisan, polimorfisma dan ikatan dinamik.

Satu pendekatan untuk mentakrif secara formal keratan untuk atur cara berorientasi objek ialah masalah utama dalam fasa penyenggaraan. Tesis ini mencadangkan takrifan keratan untuk atur cara berorientasi objek. Hubungan asas seperti Usage, Affect dan Inheritence ditakrifkan, dan mereka dipanjangkan ke kelompok hubungan kebergantungan diantara entiti-entiti dalam atur cara berorientasi objek dan mentakrif teknik mengerat berasaskan hubungan ini. Kaedah pengumpulan keratan untuk

kriteria keratan tertentu diberi juga. Pendekatan ini menunjukkan bagaimana konsep dan peraturan mengerat dapat digunakanke dalam process penyenggaraan perisian dengan memberi ilustrasi melalui contoh-contoh yang ditulis dalam bahasa Java dan Delphi.

Kajian ini juga membangunkan satu prototaip sistem alatan berorientasi objek (02SMt) yang mewakili ekstraksi secara automatik dan mengambil kebergantungan-kebergantungan sistem perisian berorientasi-objek untuk membantu dalam fasa penyenggaraan. Kebergantungan yang terjadi dan diterangkan dalam kajian ini ialah kebergantungan kawalan, kebergantungan pernyataan ke atas pembolehubah, kebergantungan pernyataan ke atas kaedah, kebergantungan pembolehubah ke atas pernyataan, kebergantungan pembolehubah ke atas kaedah, kebergantungan kelas-kelas melalui kegunaan, kebergantungan kelas-kelas melalui pewarisan, kebergantungan kelas-kelas untuk menyebabkan kesan sampingan, kebergantungankaedah ke atas kaedah lain, dan kebergantungan kaedah ke atas pembolehubah. O^2SMt mengambil keratan aturcara mengikut konsep, peraturan dan takrifan pengeratan untuk mencirikan kebergantungan dengan hubungan berorientasi-objek asas.

Kajian ini juga membincangkan graf kebergantungan berorientasi objek (O^2DG). O^2DG dikategorikan menuruti peringkat-peringkat. Kategori pertama ialah peringkat kelas yang melibatkan satu kelas kepada kelas lain. Kategori kedua ialah peringkat kaedah yang melibatkan satu kaedah atau satu pernyataan dalam satu kaedah ke kaedah lain atau pembolehubah dalam satu kelas. Kategori terakhir ialah peringkat pernyataan yang secara asasnya ialah kaedah-intra yang melibatkan pernyataan di dalam keadah yang diberi. Keratan boleh juga dikategorikan menuruti peringkat-peringkat kebergantungan yang ingin ditawan.

ACKNOWLEDGMENTS

In the name of Allah, the Beneficent, the Merciful.

I would like to take this opportunity to record my gratitude towards the many people who touched my life in significant ways; particularly those who help me during the time I was enrolling in the Ph.D. program at University Putra Malaysia.

Associate Prof. Dr. Abdul Azim Abd. Ghani and Dr. Nor Adnan Yahaya my supervisor and co-supervisor, guide and friend have shown me how wonderful a student-teacher relationship can be. They have always takes time to listen to my ideas, and they have patiently answered my questions, invaluable guidance, fruitful discussion, patience and continued encouragement provided me at every stage of this thesis.

I would like to thank all the member of my Ph.D. supervision committee. Dr. Ramlan Mahmod and Associate Prof. Dr. Md. Yazid for taking the time to review my thesis and offer their invaluable suggestions. I would also like to thank Associate Prof. Dr. Ali Mamat the member of my supervision committee during my master degree for his help and suggestions.

I would like to convey my appreciation to the Faculty of Computer Science and IT, The University Library and Universiti Putra Malaysia for providing assistance at one time or another. I would like also to thank the faculty dean Secretary Puan Norhaidah for her help and patient. My special thanks to the gentle honesty friends from Jordan, Malaysia, Iraq, Libya, … for their encouragement.

I owe very special thank to Associate Prof. Dr. Azim, Associate Prof. Dr. Bachog, and Prof. Usmani to provide me work in the Software Development Institute (SDI) my very gratefully acknowledgment to them.

My families are amongst the wisest teachers I have and they have taught me the fine arts of survival, communication, and humanity. To covert my weaknesses into my strengths; to comprehend the importance of understanding fundamental concepts in their entirety; to fully realize the harm that any form of deception can cause; to learn to adapt to new environments and coexist with a wide cross-section of humanity; to accept disappointments, rewards and appreciation; to be honest, truthful, enterprising, resourceful and hardworking. My mother, unconditional loves, prayer (Dua) and supports have been my greatest source of solace, strength and security. Gentle guidance, support, encourage and help a lot, my elder brother Ali I owe special thanks for urging me to complete my studies. I would like also record my appreciation for the help extended, support and concern shown by my brothers Khaled, Mammdoh, Mobarak, Omar and Mohd. I would like to express my warmest gratitude to my sisters, nephews, nieces, uncles, aunts, and cousins for their prayers, love, generous moral and support during my study. Finally and most important, I would like to express my most sincere and warmest gratitude to my wife for her patient and encouragement.

All praises for the Almighty, without whose will everything would cease to be.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

The last decade of the twentieth century has seen a rapid increase in the use of object-oriented approach to software development. This trend is expected to continue in this new millennium in light of the continuing progress and utilization of the Java-based technology particularly in the area of distributed computing. Even though advocates of the object-oriented approach generally believe that it can help in improving the readability of programs, the basic maintenance tasks to be carried out on them are something that still cannot be avoided. In other words, during the maintenance phase, object-oriented programs still need to be understood and later modified be it for the purpose of performing corrective or adaptive maintenance, functional enhancement as well as efficiency improvement. Therefore, if thus far various software maintenance systems have been developed to help in maintaining software systems developed through the use of the traditionally popular procedure-oriented approach, we can expect for the similar situation which, is applicable to the maintenance of object-oriented systems.

Analyzing dependencies between software components is one of the basic activities used by a maintainer to identify various relationships among program elements (Harrold and Mally 1993, Horwitz et. al. 1990, Podgurski and Lori 1990). In the case of

procedure-oriented programs, control and data dependencies are normally sufficient in helping a maintainer to understand and trace the program behavior. Although the expected benefits that one can gain through object-oriented development can be high, unfortunately maintaining object-oriented programs can be problematic if it is not done systematically. The salient features of object-oriented techniques such as polymorphism, inheritance, encapsulation, and dynamic binding are the main reasons for many maintenance problems. These additional features create additional dependencies between program elements and thus make the problem of understanding object-oriented systems more tedious and can be more complex than the procedure-oriented counterparts.

In this thesis we adapt the concept of program slicing to capture various dependencies which are useful for maintaining object-oriented programs. To support this, we propose several new concepts that form the basis for slicing object-oriented programs in general, with special attention given to those written in Java programming language. This extends the original notion of program slicing (Wieser 1979, 1982, 1984) to cover several types of program fragments that are believed to be useful in carrying out the maintenance of object-oriented programs. Finally, we propose a software tool called Object-Oriented System Maintenance tool (O^2SMt) which primarily allows the inspection of object-based relations with the help of a powerful slicing subsystem (Fawareh, and Ghani 1999B). This thesis outlines the architecture of O^2SMt and illustrates its potential use by providing several scenarios for its application in maintaining examples using Java, and Delphi programming languages.

1.2 Problem Definition

Object oriented technique is growing very fast in various applications of the computer science, programming languages, design methodologies, user interfaces databases, and operating systems. An object oriented program written for those applications may contain a lot of entity relationships which, need to be understood by the maintainers. These relationships (involving classes, message, variables, ... etc.) will make maintainers and developers face several problems to understand, make changes, modify, or enhance a huge software system that contains thousands of classes without automatic aids. Problems posed by object oriented technique include the following:

1. Software understanding.
2. Complex dependencies.
3. Inheritance, polymorphism and dynamic binding.

1. Software Understanding

Understanding a software system is a difficult problem. This is because, understand something is to know its meaning and in order to grasp the full meaning of something, one must know the reason for its existence and its nature.

There are two aspects contributing to the complex nature of software system, which are behavior and structure. The response of a system to some input is referred to as the behavior of the system. To understand the behavior of a software system means to understand the behavior of the system components and the relationships between these

components. In object oriented programs the dependencies between various components of the software are the most important to understand the behavior of the software system. Furthermore, to understand the software system for the purpose of modification and enhancement, the maintainers are required to fully understand the relations that exist within the software. The structure of a software system is determined by the logical and physical organisation of the source code and the relationships that exist within it. A thorough understanding of a software system is possible if the structure and behavior can be explained. The structure and behavior of a system are mutually dependent aspects; the structure of the system permits the software to behave in the desired way and the behavior that is expected from software is the reason as to why the software is structured in a particular way (Shrivastava 1996).

Performing a walk through various inputs and examining them can help in understanding a software system. The traces of the output of a software system is done based on complete path of execution that it follows on a particular input.

To exhaustively determine whether a software system behaves correctly (not contains any latent error) will be influenced by the input to the system. It is entirely possible for a system to behave correctly with respect to some input and not with respect to others. To exhaustively determine a software system is correct with respect to all input is impossible. Therefore any model that attempts to represent the behavior of a software system must help in understanding both kinds of behavior.

Static and dynamic relationships are two kinds of relationship that can exist between logical components of a system. A static relationship is a fixed, unchanging relationship that establishes a strong and predictable connection between components. A dynamic relationship is an indicative of a weak association between components. This kind of relationship occurs within the context of some event. An event is an occurrence that causes the system to change its configuration or state. Events cause components to associate dynamically in order to affect the change in configuration. Once the configuration has changed, the association is no longer necessary and ceases to exist. Different events cause different dynamic relationships between components. Therefore determining dynamic relationships is then based on the understanding of the events that occur in the system. The statically related components lay the groundwork for dynamic interactions to occur in a system and therefore the dynamic relationships that can be discerned from the code itself.

2. Complex dependencies

There are a lot of dependencies in an object oriented systems, such as data dependence, control dependence, calling dependence, … etc. These dependencies are represented as $X \rightarrow Y$, such that any modification in a program X must have possible effect on Y (Podgurski and Lori 1990). Furthermore, object oriented languages have special entities, such as: classes entities, methods, and messages.

The complex relationships that exist in object-oriented systems are the main cause of the dependency problem. These relationships may include indirect relations, for

example, when one class imposes indirect dependence on other classes. The traditional maintenance techniques for the structured programming are inadequate for the new object oriented programming, because they do not take into account the complexity due to special features in the unique object oriented programming. When a class is modified other classes may have to be understood and traced in order to make modification, enhancement, or correction.

3. Inheritance, polymorphism and dynamic binding

Object oriented software defines an object as an instance of a class and takes this step further to allow classes to be defined in terms of other classes. Each subclass can either inherits or override the attributes and behavior of superclass. However, subclasses are not limited to the attribute and behavior provided to them by their superclass. They can add variables and methods to the ones they inherit from the superclass. Subclasses can override inherited methods by providing specialized implementations for those methods. Subclasses provide specialized behaviors from the basis of common elements provided by the superclass. Through the use of inheritance, programmers can reuse the code in the superclass many times.

In object oriented languages, variables are used not only in the scope of the class but may also be used in other classes. The variable may be referenced by other objects of any other class using polymorphism. Furthermore, a given class can use a method, which is declared in another class, when a given message is sent to execute this method.

1.3 Aims of the Research

This research aims to provide a new set of concepts, rules and algorithms for slicing object-oriented programs to produce slices to help the maintainer in understanding these programs. The purpose is also to allow the maintainer to modify and enhance software components with the knowledge on linkages with other components.

Program slicing is known to help maintainers in understanding a foreign code. The technique in this thesis provide concepts and rules for slicing object-oriented programs according to several new dependency relations between various object oriented program entities. These dependencies according to our basic definition will help trace the impact of any proposed modifications, help in understanding object-oriented systems, and perhaps reduce the resources and efforts required for maintenance activities. The slicing concepts and rules presented are meant to cater for all object-oriented languages. Also they should work for all maintenance activities whether the maintenance is corrective, adaptive, perfective or preventive.

This research is also aimed at the developing a prototype of a software tool that provides software maintainers with a conceptual model of the architecture of the software system which is being maintained in order to help the process of understanding it. The Object Oriented System Maintenance tool (O^2SMt) allows the construction of abstract representation of object oriented software systems. The O^2SMt representation of an

object oriented software system captures basic relationships and dependencies among software components.

The object oriented dependence graph (O^2DG) is introduced to provide a graphic visualization of object oriented dependencies. This tool also provides the maintainer with an implied methodology for maintenance. The notation in this tool forms the object oriented program dependence on the sets of concepts, rules and algorithms.

The research approach gives an abstract view of the source code by capturing dependencies between software components. The dependencies between the software component are defined through a specific set of slicing concepts. Based on these, rules for slicing an object-oriented program are developed. Each rule has a special slicing algorithm. The algorithm describes the direct and indirect relationships of the software component.

In an object-oriented system, the software entities, the object oriented characteristics and relationships are specified in term of the basic constructs of the language of implementation. The presence of multiple relationships is one of the reasons for the complex dependencies within object oriented systems. The dependencies between code fragments can be determined when there is a direct and indirect relationship between the code fragments. Code fragments are related in more than one way and more than one kind of dependency between object oriented entities. Moreover different relationships may create more dependencies. From the maintenance point of view