Oriental Journal of Computer Science & Technology Vol. **3**(1), 139-144 (2010)

Static analysis and run-time coupling metrics

MANDEEP KAUR, PARUL BATRA and AKHIL KHARE

Department of Information Technology,
Bharati Vidyapeeth College of Engineering, Pune - 411 043, (India).

(Received: March 08, 2010; Accepted: April 30, 2010)

ABSTRACT

The relationships between coupling and external quality factors of object-oriented software[1] have been studied extensively for the past few years. For example, a clear empirical relationship between class-level coupling and the fault-proneness of the classes have been identified by several studies. A number of statistical techniques, principally Agglomerative Hierarchical Clustering (AHC) analysis, Byte Code Instrumentation are used to facilitate the identification of such objects. Dynamic coupling indicates the strength of association established by a connection from one software module to another at runtime. Despite the rich body of research in the field of software measurement, dynamic coupling measurement for Aspect Oriented software is still missing. A dynamic coupling measurement framework for AspectJ[10] programs is presented in this paper. The framework consists of a suite of measures for both method-level and class-level coupling relations. This paper also presents a new approach towards static analysis, in particular class analysis to the computation of dynamic coupling measures and is designed to work on incomplete programs.

Keywords: RTA, CHA, BCEL (Binary Code Engineering Library), JVM (Java Virtual Machine), Cluster Analysis.

INTRODUCTION

Using object level run time metrics to study coupling between objects" basically tells us how to identify objects from the same class that exhibit non uniform coupling behavior when measures dynamically. Coupling was first introduced in the context of the structural development techniques. Coupling is defined as, "the measure of the strength of association established by a connection from one module to another". An object of a class is coupled to another if methods of one class use methods or instance variables defined by the other.

If the coupling between modules is strong, i.e., the more inter-related they are, it is more difficult to understand, change, and correct these modules and thus making the software system more complex .A good software system should exhibit low coupling between its units.

Coupling measurement has traditionally been performed using static code analysis, because most of the existing work was done on non-object oriented code and dynamic code analysis was more expensive and complex to perform. For modern software systems, however, this focus on static analysis can be problematic because although dynamic binding existed before the advent of object-orientation, its usage has increased significantly in the last decade.

Various coupling measures are defined based on a static analysis of class code, and the ability of the CBO metric[2] to accurately predict the actual amount of coupling between objects is as yet unproven. CBO for a class is a count of the number of other classes to which it is coupled.

Statically analyzing code helps in monitoring whether it meets uniform expectations around security, reliability, performance, and maintainability[3,4,5]. Done properly, this static code analysis provides the basis for producing solid code by exposing structural errors and preventing entire classes of errors.

Traditional object-oriented coupling measures[11,12] fail to properly anticipate the quality attributes of classes and underestimate their complexity as they do not account for polymorphic interactions.

Arisholm *et al.* [6] define a certain class of dynamic coupling measures that account for polymorphism to deal with the problem of traditional object oriented coupling measures.

Dynamic Coupling Measures

Dynamic coupling measures are being collected through dynamic analyses which show that these measures are better predictors of quality attributes and better indices of complexity than traditional coupling measures.

Dynamic coupling measures are based on two well known and simple class analyses:
Class Hierarchy Analysis (CHA) and Rapid Type Analysis (RTA)[8].

Class Hierarchy Analysis *(CHA)* is the simplest form of class analysis. It combines the statically declared type of an object and the class hierarchy of the program to determine the set of possible targets of a virtual function call.

Rapid Type Analysis *(RTA)* improves on CHA by taking into account what classes are taken into account in the program.A call graph is generated in the beginning by performing Class Hierarchy Analysis. Then the information about instantiated classes is used to further reduce the executable virtual functions, thereby reducing the size of the call graph.

Metrics Data Collection Tool

A number of methods are available for collecting run-time information from Java programs. Instrumenting a Java Virtual Machine, such as JDissect, is one. A **Java Virtual Machine** (**JVM**) is a set of computer software programs and data structures that use a virtual machine model for the execution of other computer programs and scripts. The model used by a JVM accepts a form of computer intermediate language commonly referred to as Java bytecode.

Jdissect collects data from running Java programs to calculate dynamic coupling.

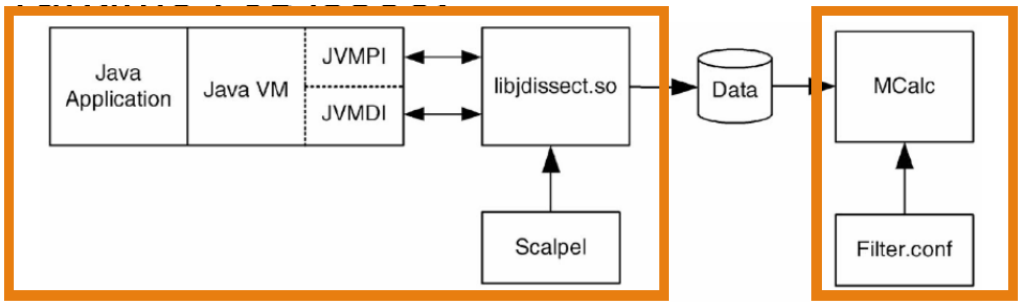

Fig. 1: JDissect

JVMPI : Java VM Profiling Interface, collecting most of the data
JVMDI : Java VM Debugging Interface, to obtain unique line number
libjdissect.so : a library of data collection routines
Scalpel : tagging to limit the scope (use case, scenario, subsystem, …)
Mcalc : read the data into data structure and analyze to obtain measures
Filter.conf : configuration file

Another technique is bytecode instrumentation in which modification of the class file content is done in order to acquire run-time information. The approach adds the least overhead to the execution of the program providing object-level accuracy, which is essential for analysis.

The BCEL (Byte Code Engineering Library) - an API can be used to analyze, create, and manipulate (binary) Java class files. All the

symbolic information of the given class, such as methods, fields and byte code instructions are in BCEL objects which represent all set of Classes. These objects can be read from an existing file, be transformed by a program and dumped to a file again.

The Case For Static Analysis

Dynamic coupling measures were captures by Arisholm *et al.*[3] through dynamic analysis; however, it was being observed that dynamic analysis has drawbacks. *First,* since it requires multiple inputs for the program for multiple executions so it is relatively slow. *Second,* the engineering task of building an instrumentation framework is relatively complex. *Third*, complete program is always required by dynamic analysis. *Fourth*, the results obtained may be incomplete as they are based on particular runs with particular inputs. For example, let there are two clients of the Bridge structure[7] in Fig. 2, one that instantiates A1 with Imp1 and one that

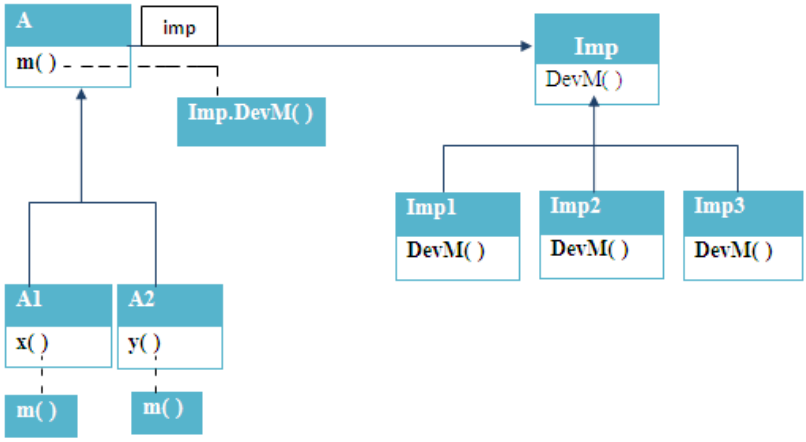

Fig. 2: Bridge structure

Instantiates A1 with Imp2. Running the two clients will count dynamic couplings 0A1,Imp1 0 and 0A1,Imp2 0 as due to call imp.DevM(). Thus, dynamic analysis will neglect four valid couplings. One would need at least six clients, and six runs, in order to capture correctly all possible dynamic couplings due to this call.

Runtime Coupling Between Objects (RCBO)

Run-time Coupling between Objects (RCBO) [2] is defined as the number of classes that are accessed by another class at run-time. This measure will be some function of the static CBO measure, as this measure determines the classes that can be theoretically accessed at run-time. This is a coarse-grained measure which will assess class-class coupling at the object level.

Advantages:

- This metric is a good predictor of the maintainability, fault proneness , testability, change proneness and reusability of a software design.

Disadvantages:

- The ability of the metric to accurately predict the actual amount of coupling between the objects is as yet unproven.
- It is defined on static analysis, thus cannot capture all the dimensions of object-level coupling, as features of object-oriented programming such as polymorphism, dynamic binding and inheritance render CBO imprecise in evaluating the run-time behavior of an application.

Run-time object level coupling basically measures the level of dependencies between objects in a system.

Run-time class level coupling measures the level of dependencies between the classes that implement the methods or variables of the caller object and the receiver object. The class of the object sending or receiving a message may be different from the class implementing the corresponding method due to the impact of inheritance.

Statistical Analysis

Coefficient of Variation (Cv)

The coefficient of variance, Cv is calculated for the RCBO results to determine how the RCBO values varies across the objects of a class. The relative scatter in data with respect to the mean is measured by Cv and is calculated by dividing the mean by the standard deviation. It has no units and can be expressed as a simple decimal value or reported as a percentage value.

When the Cv is less, the data scatter relative to the mean is small.
When the Cv is large, compared to the mean the amount of variation is large.

$$Cv = \sigma/\mu * 100 \qquad (1)$$

Equation 1 defines the coefficient of variation as a percentage.
where

μ is the mean
σ is the standard deviation.

If Cv =0 then the null hypothesis is accepted, Ho, as all objects of this classes would be accessing the same variables. However, if there was variation in the CBO values, Cv > 0, this would lead us to reject Ho and accept Hi, as the objects would be behaving differently at run-time from the point of view of coupling.

Cluster Analysis

The term *cluster analysis* (first used by Tryon, 1939)[9] encompasses a number of different algorithms and methods for grouping objects of similar kind into respective categories and can be used to discover structures in data without providing an explanation/interpretation. In other words cluster analysis is an exploratory data analysis tool which aims at sorting different objects into groups in a way that the degree of association between two objects is maximal if they belong to the same group and minimal otherwise.

The Noc value is calculated using cluster analysis. It helps reveal associations and structures of data in a domain set. A measure of proximity or similarity/dissimilarity determines groups from a complex data set. A wide variety of such measures exist but no consensus prevails over which is superior. For the purpose of this analysis, two widely used dissimilarity measures, Pearson dissimilarity and Euclidean distance, were chosen. The analysis is conducted using these two different measures in order to verify the results.

Equation 2 defines the Pearson Dissimilarity,
where μx and μy are the means of the first and second sets of data
& σx and σy are the standard deviations of the first and second sets of data.

$$d(x,y) = \frac{1/n(\Sigma I\ xi\ yi - \mu x \mu y)}{\sigma x \sigma y} \qquad (2)$$

Equation 3 defines Euclidean distance- This is probably the most commonly chosen type of distance. It simply is the geometric distance in the multidimensional space. It is computed as:

$$d(x,y) = \{\Sigma_i (x_i - y_i)^2\}^{1/2} \qquad (3)$$

The next step is an agglomerative hierarchical clustering (AHC) algorithm which provides the output related to the means of identifying coupling clusters. AHC algorithms start with singleton clusters, one for each entity. Similar pair of clusters are merged, one pair at a time, until a single cluster remains.

Throughout the cluster analysis, there is a symmetric matrix of dissimilarities maintained between the clusters. Once two clusters have been merged, it is necessary to generate the dissimilarity between the new cluster and every other cluster. The unweighted pair group average linkage

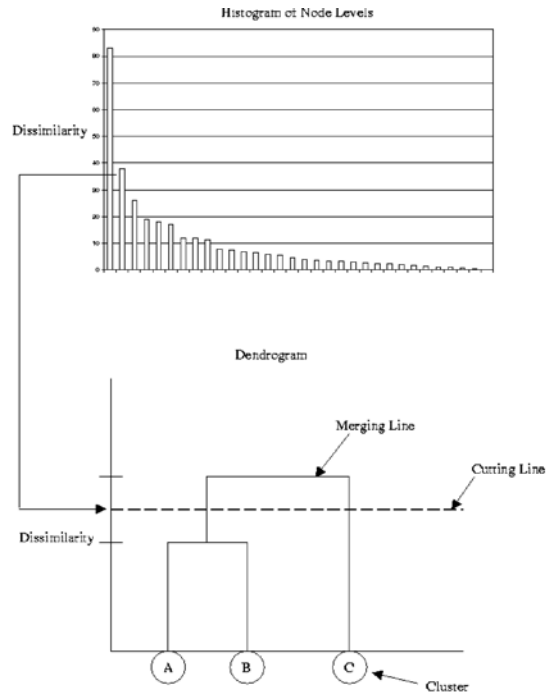

Fig. 3: Dendrogram:
At the cutting line there are two clusters

algorithm was employed here as it is theoretically the best method to use and the most likely to mimic correctly input groupings.

The output of AHC is usually represented in a special type of tree structure called a dendrogram, as illustrated by Figure 3. Each branch of the tree represents a cluster and the cutting line is a line drawn horizontally across the dendrogram at a given dissimilarity level to determine the number of clusters by constructing a histogram of node levels to find where the increase in dissimilarity is strongest, as then we have reached a level where we are grouping groups that are already homogenous. The cutting line is selected before this level is reached.

In order to accept H_0 we would expect objects from the same class to group together and occupy the same cluster, therefore expecting values of N_{OC} to be 1. The formation of a number of different clusters, where $N_{OC} > 1$, would lead us to reject H_0 and accept H_1.

DISCUSSION

The main contributions of the work are the following: First, a static analysis framework was proposed for the computation of dynamic coupling measures for strictly-typed object-oriented languages such as Java; the analysis is parameterized by a class analysis and works on incomplete programs.

Secondly, the study is done to investigate the hypothesis that from the point of view of coupling objects of a class behaves differently at run-time and for this a number of run-time object-level metrics are being used based on the static CBO measure.

ACKNOWLEDGEMENTS

The authors would like to thank Prof. A. Khare for providing the component data, and his guidance which helped to improve this paper.

REFERENCES

1. L.C. Briand, J.W. Daly, and J.K. Wust. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering,* **25**(1):91–121, (1999).
2. S.R. Chidamber and C.F. Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering,* **20**(6): 467–493, (1994).
3. V.R. Basili, L.C. Briand, and W.L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering,* **22**(10): 751–761, (1996).
4. L.C. Briand. Empirical investigations of quality factors in object-oriented software. In Empirical Studies of Software Engineering, Ottawa, Canada, (1999).

5. J. Eder, G. Kappel, and M. Schrefl. Coupling and cohesion in object–oriented systems. Technical Report 2/93, Department of Information Systems, University of Linz, Linz, Austria, (1993).
6. E. Arisholm. Dynamic coupling measurement for object-oriented software. In IEEE METRICS, pgs 33–42, (2002).
7. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, (1995).
8. D. Bacon and P. Sweeney. Fast static analysis of C++ virtual function calls. In ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, pgs 324–341, (1996).
9. S. Phattarsukol and P. Muenchaisri. Identifying candidate objects using hierarchical clustering analysis. In 8th Asia-Pacific Software Engineering Conference, pages 381–389, (2001).
10. AspectJ. http://eclipse.org/aspectj/
11. L. C. Briand, P. T. Devanbu, and W. L. Melo. An investigation into coupling measures for C++. In ACM/IEEE International Conference on Software Engineering, pgs 412–421, (1997).
12. S. Chidamber and C. F. Kemerer. Towards a metrics suite for object oriented design.In ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, pgs 197–211, (1991).