Journal of Computer Science 7 (5): 770-778, 2011
ISSN 1549-3636
© 2011 Science Publications

Design Dynamic Coupling Measurement of Distributed Object Oriented Software Using Trace Events

[1]S. Babu and [2]R.M.S. Parvathi
[1]Anna University Coimbatore and Asst. Prof, Department of IT,
Adhi Parasakthi Engineering College, Melmaruvathur, Kanchipuram, India.
[2]Sengunthar College of Engineering, Tiruchengode, Namakkal, India

Abstract: Problem statement: A common way to define and measure coupling is through structural properties and static code analysis. However, because of polymorphism, dynamic binding and the common presence of unused code in commercial software, the resulting coupling measures are imprecise as they do not perfectly reflect the actual coupling taking place among classes at run-time. For example, when using static analysis to measure coupling, it is difficult and sometimes impossible to determine what actual methods can be invoked from a client class if those methods are overridden in the subclasses of the server classes. **Approach:** Coupling measurement has traditionally been performed using static code analysis, because most of the existing work was done on non-object oriented code and because dynamic code analysis is more expensive and complex to perform. We refer to this type of coupling as dynamic coupling. In this study we propose a dynamic and efficient measurement technique over object oriented software. **Result:** We propose a hybrid model to measure the dynamic coupling present in distributed object oriented software. The proposed method has three steps; they are instrumentation process, post process and coupling measurement. First, the instrumentation process is performed. In this process, to trace method calls, a modified instrumented JVM has been used. During this process, three trace files, .prf, .clp and .svp are created. In the second step, the information present in these files, are merged. At the end of this step, the merged detailed trace of each Jvms contains pointers to the merged trace files of the other JVM's such that the path of each remote call from the client to the server can be uniquely identified. **Conclusion:** Finally, the coupling metrics are measured dynamically. The proposed system was implemented in JAVA. The implementation results show that the proposed system effectively measures the dynamic coupling.

Key words: Static code, dynamic code, object oriented, scientific method, dynamic coupling, theoretical model, structural complexity, oriented software, Java Virtual Machine (JVM)

INTRODUCTION

Software engineering describes the group of methods that construct and support software products by employing an engineering approach. Methods based on models and theories are employed by engineering disciplines. Specifying a hypothesis, designing and performing an experiment to verify its truth and interpreting the results are involved in scientific methods. Measuring the variables differentiates cases, measuring the changes in behavior and measuring the causes and effects are the supportive scientific method measurements. After the validity of a model or the truth of a theory is confirmed by scientific method, the theory is applied to practice by continuously using measurements. More visible characteristics and relationships in estimating the enormity of problems and in shaping a solution to problems can be obtained by means of effective measurements. Coupling analysis is one of the diverse methods used in software system for modeling and measuring the relationships between components. Two components having any type of connection or relationship between them are coupled by coupling analysis. Generally, the coupling nature has been categorized into diverse levels or types.

Coupling analysis attempts to capture all the attributes of the relationships between components of a given software program, by defining a theoretical model. By defining a set of measures, the coupling levels are also quantified by it. On a range of crisis that are related to the interaction among components, the theoretical model and the measurement set serve as a

Corresponding Author: S. Babu, Research Scholar, Anna University Coimbatore and Asst. Prof Department of IT, Adhi parasakthi Engineering College, Melmaruvathur, Kanchipuram

foundation for implementing complexity analysis. A major role is played by software metrics in the planning and control of software development projects. Software development and maintenance has important applications for coupling measures. In software, the causes for structural complexity are explained and quality attributes such as fault-proneness, ripple effects of changes and changeability are predicted by coupling measures. The extent to which each program module depends on each one of the other modules is termed as coupling or dependency. Static usage dependencies between the classes in an object-oriented system are portrayed by coupling measures. "Static" couplings only are taken into account by conventional coupling measures. They may considerably underestimate the complexity of software leading to underestimation of code inspection, testing and debugging needs because dynamic coupling due to polymorphism are not taken into account. Therefore, inferior predictive accuracy is likely in quality models that utilize static coupling measurement.

Works available in the literature for software metrics have mainly concentrated on centralized systems and only very few of them have focused on distributed systems and more specifically on service-oriented systems. Conventional non distributed systems differ from systems with distributed components in several ways including communication type, latency, concurrency, partial, versus total failure and referencing parameters- passing strategies. Normally, distributed systems with service oriented components are more complex because they accomplish efficiency and other quality characteristics in a more heterogeneous networking and implementation environment. The importance of software quality has been accepted a matter of great concern not only for the developers but also for the business and government customers.

As it was well-Known before, quality depends mainly upon the maintainability of the software. Coupling measurement is undoubtedly one of the benchmarking methods whether the ready-to-launch application has reliable maintainability or not. The measurement, in one hand, has traditionally been performed simply using static code analysis at the stage of system testing or sometimes at the trial operation stage. The static analysis is an appropriate measure while we were using traditional programming languages like COBOL, FORTRAN, Pascal and C among others. With the crowded popularity of Object-Oriented (OO) languages like C++, Visual C, Java and applications implemented in those OO languages, on the other hand, the dynamic coupling measure could be used for the evaluation of the systems and application. One of the main reason to apply dynamic coupling metrics is that it can reflects the reality of the tested application since the measure could pinpoint the pitfalls and shortages that are not to be expected to be found with static coupling measure although dynamic measure takes much more time to quality-test especially if the size of the subject application is large.

In the context of object-oriented systems, research related to quality models has focused mainly on defining structural metrics (e.g., capturing class coupling) and investigating their relationships with external quality attributes (e.g., class fault-proneness) (Chidamber *et al*., 2005). The ultimate goal is to develop predictive models that may be used to support decision making, e.g., decide which classes should undergo more intensive verification and validation. Regardless of the structural attribute considered, most metrics have been so far defined and collected based on a static analysis of the structural attribute considered, most metrics have been so far defined and collected based on a static analysis of external quality attributes, such as fault-proneness (Briand and Labiche, 2002), ripple effects after changes (Briand *et al.*, 1999; Kabaili *et al*., 2001) and changeability (Arisholm, 2001; 2002; Arisholm *et al*., 2001, Aly and Abuelnasr, 2010). However, many of the systems that have been studied showed little inheritance and, as a result, limited use of polymorphism and dynamic binding (Deligiannis *et al*., 2002). As the use of object-oriented design and programming matures in industry, we observe that inheritance and polymorphism are used more frequently to improve internal reuse in a system and facilitate maintenance. Though no formal survey exists on this matter, this is visible when analyzing the increasing number of open source projects, application frameworks and libraries. The problem is that the static, coupling measures that represent the core indicators of most reported quality models (Briand and Labiche, 2002) lose precision as more intensive use of inheritance and dynamic binding occurs. This is expected to result in poorer predictive accuracy of the quality models that utilize static coupling measurement.

A common way to define and measure coupling is through structural properties and static code analysis. However, because of polymorphism, dynamic binding and the common presence of unused ("dead") code in commercial software, the resulting coupling measures are imprecise as they do not perfectly reflect the actual coupling taking place among classes at run-time. For example, when using static analysis to measure coupling, it is difficult and sometimes impossible to determine what actual methods can be invoked from a client class if those methods are overridden in the subclasses of the server classes.

Coupling measurement has traditionally been performed using static code analysis, because most of the existing work was done on non-object oriented code and because dynamic code analysis is 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. We refer to this type of coupling as dynamic coupling. An empirical evaluation of the proposed dynamic coupling measures is reported in which we study the relationship of these measures with the change proneness of classes. Preliminary results suggest that some dynamic coupling measures are significant indicators of change proneness and that they complement existing coupling measures based on static analysis.

Classification of coupling measures: Existing coupling measures can be broadly classified into the following two groups:

- Procedural programming coupling measures: these measure the coupling of software components that are implemented in procedural programming languages; examples include metrics proposed by); Briand *et al*. (1999); Cartwright and Shepperd (2000); Chaumun *et al*. (2000) and Chidamber and Kemerer (2005). This class of metrics is heavily influenced by the classification of coupling levels.

- Object-oriented coupling measures : these measure the coupling of software components that are implemented in object-oriented programming languages; examples include metrics prop osed by Chidamber *et al*. (2005); Deligiannis *et al*. (2002); and Freund and Wilson (1998)

Existing evidence suggests that dynamic coupling could be of strong interest. A preliminary empirical study on a Smalltalk system suggests that there is a significant relationship between change proneness and dynamic coupling (Arisholm, 2002). Furthermore, according to the results of a controlled experiment (Arisholm *et al*., 2001), static coupling measures may sometimes be inadequate when attempting to explain differences in changeability (e.g., change effort) for object-oriented designs. A follow-up study indicates that the actual flow of messages taking place between objects at run-time is often traced systematically by professional developers when attempting to understand object-oriented software (Kabaili *et al*., 2001). The results thus suggest that dynamic coupling measures could be of interest as predictors of the cognitive complexity of object-oriented software. Finally, dynamic coupling is more precise than static coupling for systems with dead (unused) code, which is uninteresting in most situations and can seriously bias analysis.

Traditional coupling measures take into account only "static" couplings. They do not account for "dynamic" couplings due to polymorphism and may significantly underestimate the complexity of software and misjudge the need for code inspection, testing and debugging. This is expected to result in poorer predictive accuracy of the quality models in Distributed Object-Oriented System that utilize static coupling measurement.

We first distinguish different types of dynamic coupling measures. Then, based on this Classification, we provide both informal and formal definitions, using a working example to illustrate the fundamental principles. Our measures were designed to fulfill five properties that we deem very important for any coupling measure to be well formed. In order to define measures in a way that is programming language independent, we refer to a generic data model defined with a UML class diagram.

Entity of measurement: Since dynamic coupling is based on dynamic code analysis, coupling may be measured for a class or one of its instances. The entity of measurement may therefore be a class or an object.

Granularity: Orthogonal to the entity of measurement; dynamic coupling measurement can be aggregated at different levels of granularity (Table 1). With respect to dynamic object coupling, measurement can be performed at the object level, but can also be aggregated at the class level, i.e., the dynamic coupling of all instances of a class is aggregated. In practice, even when measuring object coupling, the lowest level of granularity is likely to be the class, as it is difficult to imagine how the coupling measurement of objects could be used. Alternatively, all the dynamic coupling of objects involved in an execution scenario can be aggregated. We can also measure the dynamic object coupling in entire use cases (i.e., sets of scenarios), sets of use cases, or even an entire system (all objects of all use cases). In the case where the entity of measurement is a class, the aggregation scale is different as we can aggregate dynamic coupling across an inheritance hierarchy, a subsystem, a set of subsystems, or an entire system.

Scope: Another important source of variation in the way we can measure dynamic coupling is the scope of measurement. This determines which objects or classes, depending on the entity of measurement, are to be accounted for when measuring dynamic coupling. For example, we may want, depending on the application context, to exclude library and framework classes. Simula At the level, we may want to exclude certain use cases modeling exceptional situations (e.g., error conditions, usually modeled as extended use cases (Lakhotia and Deprez, 1999) or objects that are instances of library or framework classes. At the very least, we may want to distinguish the different types of coupling taking place in these different categories. The choices we make regarding the entity, granularity and scope of measurement depend on how we intend to apply dynamic coupling.

MATERIALS AND METHODS

Collecting dynamic coupling data at distributed environment: It is crucial to collect dynamic coupling data in a practical and efficient manner, Based on dynamic UML models.

We propose a hybrid model to measure the dynamic coupling present in distributed object oriented software. The proposed method has three steps; they are:

- Instrumentation process
- Post process
- Coupling measurement

First, the instrumentation process is performed. In this process, to trace method calls, a modified instrumented JVM has been used. During this process, three trace files, .prf, .clp and .svp are created. In the second step, the information present in these files, are merged. At the end of this step, the merged detailed trace of each Jvms contains pointers to the merged trace files of the other JVM’s such that the path of each remote call from the client to the server can be uniquely identified. Finally, the coupling metrics are measured dynamically. The proposed system was implemented in JAVA. The implementation results show that the proposed system effectively measures the dynamic coupling.

Collecting distributed dynamic coupling measures at runtime in the distributed environment: To collect dynamic coupling data from Java applications, we developed a method: Trace Event. An overview of the architecture is depicted in Fig. 1. The method separates the collection and analysis of dynamic coupling data into three phases. In the first phase is instrumentation process, in this process we are using trace event method to trace the .prf, .clp and .svp from a running Java program is gathered and stored. This is accomplished by having the Java Virtual Machine (JVM) load a library of data collection routines that are called whenever specified internal events occur. The interfaces used for communication between the JVM and the library are called JVMPI (Java VM Profiling Interface) and JVMDI (Java VM Debugging Interface). Most of the data is collected from the profiling interface. The JVMDI is used to obtain the unique line number from which a method call originates (to obtain the information needed to calculate the measures).

During the instrumentation process phase, a user may interactively tag messages belonging to specific scenarios or use cases through a separate utility that communicates with distributed systems so through a socket connection. These tags can subsequently be used to limit the scope of measurement (e.g., to specific use cases) and, potentially, to compute measures at higher levels of granularity than the class (e.g., at the use case aggregation level). When the application terminates, the data is stored in a flat file structure (Data).

In the second phase, the information present in these files like .prf, .clp and, .svp are merged and the data is analyzed.

In the third phase, the merged detailed trace of each Jvms contains pointers to the merged trace files of the other JVM’s such that the path of each remote call from the client to the server can be uniquely identified. Finally, the coupling metrics are measured dynamically. The proposed system was implemented in JAVA. The implementation results show that the proposed system effectively measures the dynamic coupling.

Each measure is then computed simply by counting the number of elements in each set. Data from several runtime sessions can be merged by the analysis tool, such that accumulated dynamic coupling data can be computed. This merging capability enables the collection of coupling data for Java systems for which several concurrent instances of the JVM are used, such as large, distributed, or component-based systems.

Our coupling method utilizes interfaces provided by the Java Virtual Machine to collect the message traces and other information. Instrumentation can be done either at the source code or byte code level using tools such as the Java Compiler Compiler (JavaCC) (Java.net, 2003) or the Byte Code Engineering Library (BCEL) (Jakarta, 2003), respectively. However, utilizing the existing interfaces to the Java VM provides several benefits over instrumentation. Instrumenting the code means that we are testing the instrumented version and not the actual version, which may lead to different outputs and system states. Since instrumentation causes a significant effort overhead, if the system is evolving rapidly, the project manager will also be reluctant to keep instrumenting the new versions.

Furthermore, source code instrumentation requires access to the Java application source code. This might be a disadvantage in cases where an application uses libraries for which the source code is not available. Finally, instrumentation might cause a significant performance overhead. In contrast to our approach, both source code and byte code instrumentation require that parts of the data collection software be written in Java. Subsequently, the byte code of the data collection software is interpreted by the Java VM. Since our data collection tool is written in C++ and dynamically linked with the JVM at runtime, there is probably less performance overhead associated with our approach than with data collection tools employing instrumentation. As performance overhead increases, the behavior of concurrent software is more likely to be affected by the data collection process and it is important to minimize the chances of such a problem occurring.

Working example: We now use a small working example, as shown in Fig. 1, though it is assumed that our measures are collected through static and dynamic analysis of code, we use UML to describe a fabricated example, because it is more legible than pseudo code. This example is designed to illustrate the subtleties arising from polymorphism and dynamic binding. Other aspects, such as method signatures, have been intentionally kept simple to focus on polymorphism and dynamic binding.

The following sets can be derived from above Fig. 2:

Class C = {c1, c2, c3, c4, c5}
Method M = {m1, m2, m3}
RMC= {(m1, c1), (m2, c2), (m3, c3)}:

Definitions of measures: The measures are all defined as cardinalities of specific sets. They are therefore defined on an absolute scale and are amenable, as far as measurement theory is concerned; to the type of regression analysis performed. First, as mentioned above, we differentiate the cases where the entity of measurement is the object or the class. Second, as in previous static coupling frameworks (Briand *et al*., 1999), we differentiate import from export coupling, that is the direction of coupling for a class or object. For example, we differentiate whether a method executed on an object calls (imports) or is called by (exports) another object's method. Furthermore, orthogonal to the entity of measurement and direction of coupling considered, there are at least three different ways in which the strength of coupling can be measured. First, we provide definitions for import and export coupling when the entity of measurement is the object and the granularity level is the class. Phrases outside and between parentheses capture the situations for import and export coupling, respectively.

Dynamic messages: Within a runtime session, it is possible to count the total number of distinct messages sent from (received by) one object to (from) other objects, within the scope considered. That information is then aggregated for all the objects of each class. Two messages are considered to be the same if their source and target classes, the method invoked in the target class and the statement from which it is invoked in the source class are the same. The latter condition reflects the fact that a different context of invocation is considered to imply a different message. In a UML sequence diagram, this would be represented as distinct messages with identical method invocations but different guard conditions.

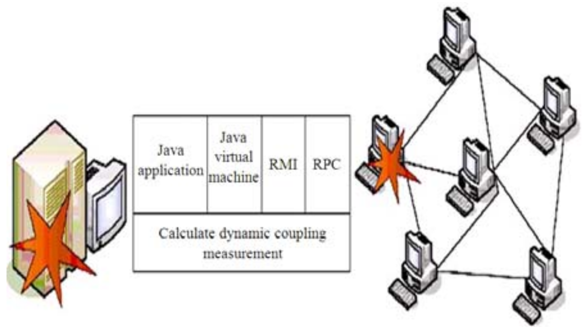


Fig. 1: Dynamic coupling data at distributed environment

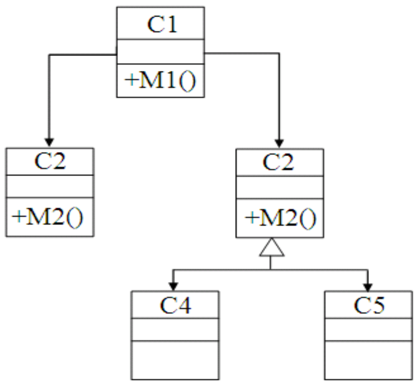


Fig. 2: Class diagram example (UML notation)

Table 1: Dynamic coupling classification

S. No	Entity	Granularity aggregation level	Scope (Include/Exclude)
1	Class	Class, inheritance Hierarchy, Systems and Set of sub systems	Library files and classes, Framework files and classes
2	Object	Class, object, set of use cases, set of scenarios and systems	Library objects, Framework objects, Exceptional use cases

Distinct method invocations: A simpler alternative is to count the number of distinct methods invoked by each method in each object (that invokes methods in each object). Note that this is different from simply counting method invocations as we count each distinct method only once. That information is then aggregated for all the objects of each class.

Distinct classes: It is also possible to count only the number of distinct server (client) classes that a method in a given object uses (is used by). That information is then aggregated for all the objects of each class.

Analysis of properties: We show here that the five coupling properties presented in (Briand *et al*., 1999) are valid for our dynamic coupling measures. The motivation is to perform an initial theoretical validation by demonstrating that our measures have intuitive properties that can be justified.

Nonnegativity: It is not possible for the dynamic coupling measures to be negative because they measure the cardinality of sets, e.g., IC OM returns a set of tuples (m, c, m', c') 2 M_ C _M_ C.

Null values: At the system level, if S is the set that includes all the objects that participate in all the use cases of the system, IC (M_S) is empty (and coupling equal to 0) if and only if the set of messages in S is empty.

Monotonicity: If a class c is modified such that at least one instance o sends/receives more messages, its import/export coupling can only increase or stay the same, for any of the coupling measures.

Impact of merging classes: Assuming c0 is the result of merging c1 and c2, thus transforming system S into S0, for any Coupling measure, we want the following properties to hold at the class and system levels:

Coupling (c1)+ Coupling(c2) > = Coupling(c)
Coupling (S) > = Coupling(S')

Merging uncoupled classes: Following reasoning similar to that above, if two classes' c1 and c2 do not have any coupling, this means there is no tuple of the type (m1, c1, m2, c2) in IV. If we merge them into one class, we therefore cannot obtain tuples of the type (m1, c0, m2, c0).

Related work: Arisholm *et al*. (2003) defined and validated a number of dynamic coupling metrics and studied the relationship of these with the change proneness of a system. They found that the dynamic coupling measurement did capture additional properties to the static coupling metrics and were good predictors of the change proneness of a class. Chidamber and Kemerer (2005) originally defined CBO for a class as "a count of the number of non inheritance related couples with other classes. An object of a class is coupled to another if methods of one class use methods or instance variables defined by the other. They later revised their definition to state (Thwin and Quah, 2003), "CBO for a class is a count of the number of other classes to which it is coupled.

Briand *et al*. (1999) carried out an extensive survey of the available literature on coupling in object-oriented systems and concluded that all the metrics at that time measured coupling statically, at the class level. No measures of runtime object level coupling had been proposed.

Yacoub *et al*. (1999) described a set of dynamic coupling metrics designed to evaluate the change-proneness of a design. The metrics were applied at the early development phase to determine design quality. They used executable object-oriented design models to model the application to be tested. The metrics were evaluated for a number of different execution scenarios and they extended the scenarios to have an application scope.

Existing literature on software metrics is mainly focused on centralized systems. Yacoub *et al*. (1999) while work in the area of distributed systems, particularly in service-oriented systems, is scarce. Systems with distributed components differ from traditional non distributed systems along a number of dimensions including communication type, latency, concurrency, partial versus total failure and referencing/parameter-passing strategies. Distributed systems with service-oriented components are even more complex, since efficiency and other quality attributes must be achieved in a typically more heterogeneous networking and execution environments. Object-oriented analysis and design are popular concepts in today's software development environment. They are often heralded as the silver bullet for solving software problems, while in reality there is no silver bullet; object-oriented has proved its value for systems that must be maintained and modified. Object-oriented software development requires a different approach from more traditional functional decomposition and data flow development methods. While the functional and data analysis approaches commence by considering the systems behavior and/or data separately; object-oriented analysis approaches the problem by looking or system entities that combine them. Object-oriented analysis and design focuses on objects as the primary agents involved in a computation; each class of data and related operations are collected into a single system entity. The concepts of software metrics are well established and many metrics relating to product quality have been developed and used. For evaluating software quality that has four goals

- Stability o f requirements and design
- Product quality
- Testing effectively and
- Implementation effectively

With object-oriented analysis and design methodologies gaining popularity, it is time to start investigating object-oriented metrics with respect to these goals.

RESULTS

The coupling metrics for Distributed Object Oriented System, which is proposed in this paper, was implemented in the working platform of JAVA (version JDK 1.6). The results obtained from the proposed method are described as follows.

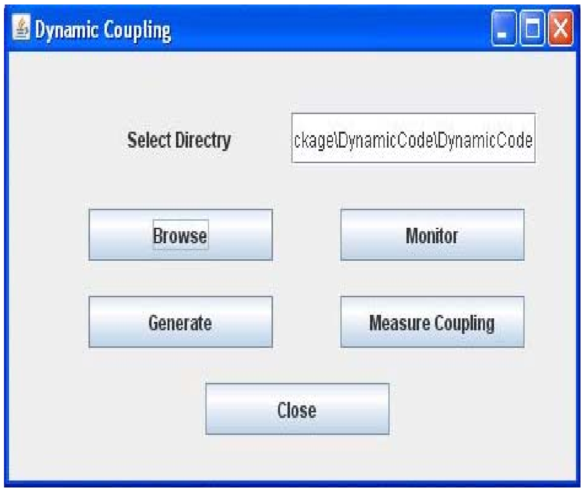


Fig. 3: Initial screen obtained in the proposed system

When we execute the proposed method, the initial screen obtained which is described in Fig. 3. In this, the browse button is used to select the package. In Fig. 4, the instrumentation process is described.

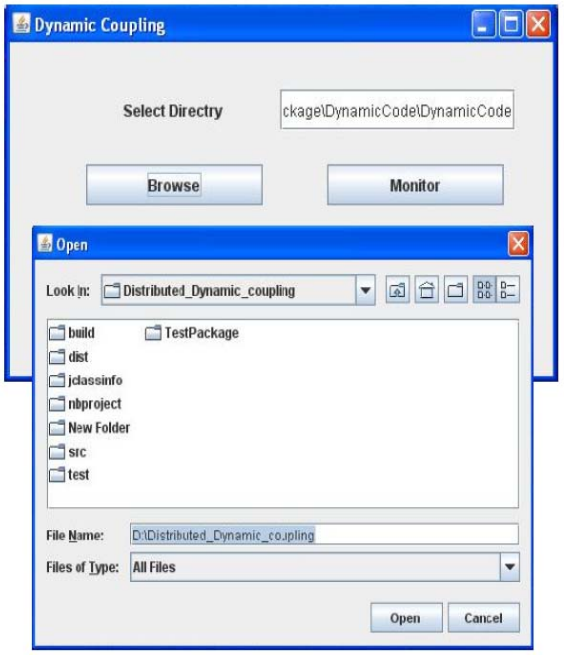
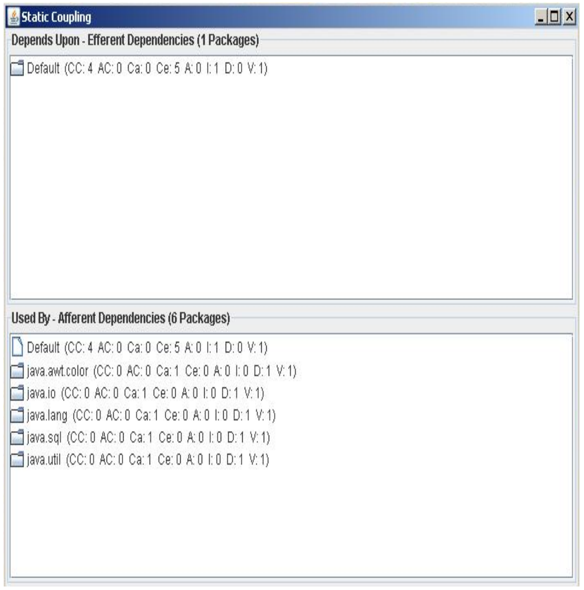


Fig. 4: The sample output obtained in the Instrumentation process



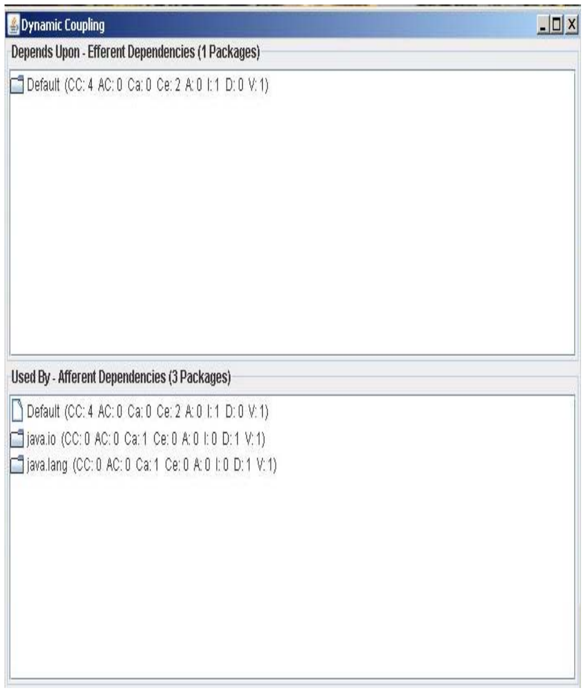
Depends Upon - Efferent Dependencies (1 Packages)

```
Default (CC: 4 AC: 0 Ca: 0 Ce: 5 A: 0 I: 1 D: 0 V: 1)
```

Used By - Afferent Dependencies (6 Packages)

```
Default (CC: 4 AC: 0 Ca: 0 Ce: 5 A: 0 I: 1 D: 0 V: 1)
java.awt.color (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1 V: 1)
java.io (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1 V: 1)
java.lang (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1 V: 1)
java.sql (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1 V: 1)
java.util (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1 V: 1)
```

Fig. 5: The static coupling measurements


Dynamic Coupling
Depends Upon - Efferent Dependencies (1 Packages)
Used By - Afferent Dependencies (3 Packages)

```
Default (CC: 4 AC: 0 Ca: 0 Ce: 2 A: 0 I: 1 D: 0 V: 1)
```

```
Default (CC: 4 AC: 0 Ca: 0 Ce: 2 A: 0 I: 1 D: 0 V: 1)
java.io (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1 V: 1)
java.lang (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1 V: 1)
```

Fig. 6: The dynamic coupling measurements

These Fig. 5 and 6 show, the number of packages used in both static and dynamic coupling. In this, the packages are used by both client and servers.

DISCUSSION

In this paper, we have proposed a new approach to the computation of dynamic coupling measures in DOO systems by introspection and adding trace events into methods. First, we provide formal, operational definitions of coupling measures and analysis. We propose dynamic coupling measures for distributed object-oriented systems i.e., coupling measurement on both clients and server dynamically. We described the classification of dynamic coupling measures. The motivation for those measures is to complement existing measures that are based on static analysis by actually measuring coupling at runtime in the hope of obtaining better decision and prediction models because we account precisely for inheritance, polymorphism and dynamic binding. Admittedly, many other applications of dynamic coupling measures can be envisaged. However, investigating change proneness was used here to gather initial but tangible evidence of the practical interest of such measures. Finally we propose our dynamic coupling measurement techniques which involve Introspection Procedure, Adding trace events into methods of all classes and Predicting Dynamic Behavior while running the source code. The source code is filtered to arrive the Actual Runtime used Source Code which is then given for any standard coupling technique to get the Dynamic Coupling.

CONCLUSION

In the above we have discussed about the distributed object oriented system for coupling measurement, in future we have to analyses some steps to showcase that our proposed scheme behaves efficiently than the existing one. We are analyzing the various Coupling Measurement in Object Orient Software and propose the Hybrid model in Distributed Object Oriented (DOO) Software for dynamic coupling measurement.

REFERENCES

Arisholm, E., 2001. Empirical assessment of changeability in object-oriented software. PhD Thesis, Dept. of Informatics, Univ. Oslo, ISSN: 1510-7710.

Arisholm, E., 2002. Dynamic coupling measures for object-oriented software. Proceeding of the 8th IEEE Symposium Software Metrics (METRICS '02), IEEE Computer Society Washington, DC, USA., pp: 33-42. ISBN: 0-7695-1339-5

Arisholm, E., D.I.K. Sjoberg and M. Jorgensen, 2001. Assessing the changeability of two object-oriented design alternatives-a controlled experiment. Empirical Software Eng., 6: 231-277. DOI: 10.1023/A:1011439416657

Arisholm, E., L.C. Briand and A. Foyen, 2003. Dynamic coupling measurement for object-oriented software. Technical Report, Simulation Research Laboratory, http://www.simula.no/~erika.

Briand, L.C. and Y. Labiche, 2002. A UML-based approach to system testing. Software Syst. Model., 1: 10-42. DOI: 10.1007/s10270-002-0004-8

Briand, L.C., J. Wust and H. Lounis, 1999. Using coupling measurement for impact analysis in object-oriented systems. Proceeding of the International Conference Software Maintenance (ICSM '99), IEEE Computer Society Washington, DC, USA., pp: 475-482. SBN:0-7695-0016-1

Briand, L.C., J.W. Daly and J. Wust, 1999. A unified framework for coupling measurement in object-oriented systems. IEEE Trans. Software Eng., 25: 91-121. ISSN: 0098-5589

Cartwright, M. and M. Shepperd, 2000. An empirical investigation of an object-oriented software system. IEEE Trans. Software Syst., 26: 786-796. ISSN: 0098-5589

Chaumun, M.A., H. Kabaili, R.K. Keller, F. Lustman and G. Saint- Denis, 2000. Design properties and object-oriented software changeability. Proceeding of the 4th Euromicro Working Conference Software Maintenance and Reeng., Feb. 29- Mar. 03, Zurich, Switzerland, pp: 45-54. ISBN: 0-7695-0546-5

Chidamber, S.R., D.P. Darcy and C.F. Kemerer, 2005 Springer Science + Business Media. Inc. Manufactured in The Netherlands.

Deligiannis, I.S., M. Shepperd, S. Webster and M. Roumeliotis, 2002. A review of experimental investigations into object-oriented technology. Empirical Software Eng., 7: 193-231. DOI: 10.1023/A:1016392131540

Emam, K.E., S. Benlarbi, N. Goel and S.N. Rai, 2001. The confounding effect of class size on the validity of object-oriented metrics. IEEE Trans. Software Eng., 27: 630-650. ISSN: 0098-5589

Aly, W.M. and M.S. Abuelnasr, 2010. Electronic design automation using object oriented electronics. *Am. J. Eng. Applied Sci. 3: 121-127*. DOI: 10.3844/ajeassp. 2010.121.127

Freund, R.J. and W.J. Wilson, 1998. Regression Analysis: Statistical Modeling of a Response Variable. 2nd Edn., Academic Press, ISBN0120885972, pp: 459.

Jakarta, 2003. The apache Jakarta project. http://jakarta.apache.org/

Java.net, 2003. Java Compiler Compiler (JavaCC). https://javacc.dev.java.net/

Kabaili, H., R. Keller and F. Lustman, 2001. Cohesion as changeability indicator in object-oriented systems. Proceeding of the IEEE Conference Software Maintenance and Reengineering (CSRM'01), IEEE Computer Society Washington, DC, USA., pp: 39-46. ISBN: 0-7695-1028-0

Lakhotia, A. and J.-C. Deprez, 1999. Restructuring functions with low cohesion. Proceeding of the IEEE Working Conference Reverse Engineering (WCRE'099), IEEE Computer Society Washington, DC, USA., pp: 36-46. ISBN:0-7695-0303-9

Thwin, M.M.T. and T.-S. Quah, 2003. Application of neural networks for software quality prediction using object-oriented metrics. Proceeding of the IEEE International Conference Software Maintenance (ICSM'03), Sep. 22-26, Amsterdam, The Netherlands. ISBN: 0-7695-1905-9

Yacoub, S.M., H.H. Ammar and T. Robinson, 1999. Dynamic metrics for object-oriented designs. Proceeding of the IEEE 6th International Symposium Software Metrics (Metrics '99), Nov. 04-06, Boca Raton, FL , USA., pp: 50-61. ISBN: 0-7695-0403-5

Yacoub, S., H. Ammar and T. Robinson, 2000. A methodology for architectural-level risk assessment using dynamic metrics. Proceeding of the 11th International Symposium Software Reliability Engineering, Oct. 08-11, San Jose, California, pp: 210-221. ISBN: 0-7695-0807-3