Development of Dynamic Coupling Measurement of Distributed Object Oriented Software Based on Trace Events

S. Babu[1] and Dr.R.M.S. Parvathi[2]

[1]Research Scholar, Anna University of Technology, Coimbatore, India.

babubalaji2k5@gmail.com

[2]Principal, Sengunthar College of Engineering, Tiruchengode, Namakkal, India.

rmsparvathi@india.com

ABSTRACT

Software metrics are increasingly playing a central role in the planning and control of software development projects. Coupling measures have important applications in software development and maintenance. Existing literature on software metrics is mainly focused on centralized systems, while work in the area of distributed systems, particularly in service-oriented systems, is scarce. Distributed systems with service oriented components are even more heterogeneous networking and execution environment. 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 poor predictive accuracy of the quality models in distributed Object Oriented systems that utilize static coupling measurements. In order to overcome these issues, we propose a hybrid model in Distributed Object Oriented Software for measure the coupling dynamically. In the proposed method, there are three steps such as Instrumentation process, Post processing and Coupling measurement. Initially the instrumentation process is done. In this process the instrumented JVM that has been modified to trace method calls. During this process, three trace files are created namely .prf, .clp, .svp. In the second step, the information in these file are merged. At the end of this step, the merged detailed trace of each JVM contains pointers to the merged trace files of the other JVM such that the path of every remote call from the client to the server can be uniquely identified. Finally, the coupling metrics are measured dynamically. The implementation results show that the proposed system will effectively measure the coupling metrics dynamically.

KEYWORDS

Distributed Object Oriented (DOO) Systems, Software Engineering, Dynamic coupling, Static coupling, Instrumentation, Trace events.

1. INTRODUCTION

Software engineering is an engineering discipline that is concerned with all aspects of software production. Software products consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability. The software process consists of activities that are involved in developing software products. Basic activities are software specification, development, validation and evolution. Methods are organized ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design

DOI : 10.5121/ijsea.2012.3112

guidelines. Object-oriented technology is built upon a sound engineering foundation, whose elements are collectively called the object model [1]. This model encompasses many useful software construction features such as abstraction, encapsulation, modularity, inheritance, typing, genericity and dynamic binding. Therefore, object model is useful for understanding problems, communicating with application experts and modeling complex enterprises into a software design [2]. This technology also helps to promote software reusability, maintainability, reliability and performance [3].The popularity of the internet coupled with advances in local area network and high speed network technologies have introduced many new distributed applications. Examples include software in the area of computer supported collaborative work, airline and hotel reservation systems, and banking systems - to name just a few. Object-oriented techniques are often used to cope with the complexity of developing these software systems which have come to be known as distributed object-oriented software systems (DOOSS) [4].

Distributed systems have also become increasingly common as more and more organizations use networks to share resources, enhance communication and increase performance. Examples of these systems range from the Internet, to workstations in a local area network within a building, to processors within a single multiprocessor [5]. In a distributed object-oriented application, classes can run on a separate computer within a network system. So, they should be distributed efficiently among different nodes. [6]. A distributed OO application consists mainly of a set of interacting objects; each one runs on a separate computer within a network system. There have been a large number of projects related to the use of object-oriented approaches for the design of problem solving environments for complex applications in various scientific fields [7]. The object model and distributed technologies are being amalgamated [8]. The advantage is obvious: the complexity and dependencies of the entities can make use of the object model in a distributed system to break down the intensive design process into efficient constructs. Many of the concepts of object-oriented programming are currently finding widespread application in loosely coupled distributed systems.

They all adopt the philosophy of a 'uniform' or 'virtual' object space which hides the disjoint address spaces in the network and permits all entities in the system, from the highest level application layers to the lowest level system layers, to be modeled as objects [9]. Moreover, the advanced techniques contributed by COBRA, OLE and WWW have greatly promoted the approach of distributed object technology [10]. In computer science, coupling or dependency is the degree to which each program module relies on each one of the other modules. Coupling is defined as, “the measure of the strength of association established by a connection from one module to another” [11]. Coupling can be defined and precisely measured based on dynamic analysis of systems. We refer to this type of coupling as dynamic coupling [12]. Coupling analysis is a powerful tool for analyzing interactions among components in software. In coupling analysis, two components are coupled if a connection or relationship exists between them. The coupling nature is categorized into different levels. Coupling analysis tries to capture all the attributes of software that are related to relationships among components by defining a theoretical model, and to quantify the coupling levels by defining a set of measures. The theoretical model and the measurement set serve as a foundation for exercising complexity analysis on various problems that are related to the interaction among components. A large number of coupling measures have been defined for object-oriented (OO) systems. [13].

Coupling measures have important applications in software development and maintenance. They are used to help developers, testers and maintainer’s reason about software complexity and software quality attributes [14]. The current research on modeling and measuring the relationships among software components through coupling analysis is insufficient. Coupling measures are incomplete in their precision of definition and quantitative computation. In

particular, current coupling measures do not reflect the differences in and the connections between design level relationships and implementation level connections. Hence, the way coupling is used to solve problems is not satisfactory. 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. Due to inheritance, the class of the object sending or receiving a message may be different from the class implementing the corresponding method.

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 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. Static analysis cannot capture all the dimensions of object-level coupling, as features of object-oriented programming such as polymorphism, dynamic binding and inheritance in evaluating the run-time behavior of an application [10]. As per the previous research and experiments, dynamic coupling is more precise than static coupling for systems with unused code [15].

In the proposed system, we are going to calculate the coupling measures dynamically in a Distributed Object Oriented (DOO) system. Dynamic coupling in DOO system means, calculating the coupling dynamically on both clients and server. The rest of the paper is organized as follows: Section 2 deals with some of the recent research works related to the proposed technique. Section 3 explains the Distributed Object Oriented Systems. Section 4 describes the proposed technique for coupling measurement with all necessary mathematical formulations and figures. Section 5 discusses about the experimentation and evaluation results with necessary tables and graphs and section 6 concludes the paper.

2. A SURVEY OF RECENT RESEARCH IN THE FIELD

Kai Qian *et al* [16] have presented a service decoupling metrics for service-oriented distributed software composition. The proposed metrics can be applied in the selection of service component in the service-oriented software design process and to evaluate the service-oriented software as a whole in term of decoupling quality attribute for better software understandability, maintainability, reliability, testability, and reusability.

Benjamin A. Allan *et al*. [17] have described the common component architecture (CCA) which provides a means for software developers to manage the complexity of large scale scientific simulations and to move toward a plug and play environment for high performance computing. In the scientific computing context, component models also promote collaboration using independently developed software, there by allowing particular individuals or groups to focus on the aspects of greatest interest to them. The CCA supports parallel and distributed computing as well as local high performance connections between components in a language-independent manner. The design places minimal requirements on components and thus facilitates the integration of existing code into the CCA environment. The CCA model imposes minimal overhead to minimize the impact on application performance. The focus on high performance distinguishes the CCA from most other component models. The CCA was being applied within

an increasing range of disciplines, including combustion research, global climate simulation, and computational chemistry.

Liguo Yu [18] had presented a method to correlated evolutionary coupling and reference coupling. They studied the evolution of 597 consecutive versions of Linux and measure the evolutionary coupling and reference coupling among 12 kernel modules. They compared 12 pairs of evolutionary coupling data and reference coupling data. The results showed that linear correlation existed between evolutionary coupling and reference coupling. They concluded that in Linux, the dependencies between software components induced via the system architecture had noticeable effects on kernel module co-evolution.

Safwat H. Hamad *et al*. [19] have presented an approach for efficiently restructuring the Distributed Object Oriented software classes in order to be mapped onto target distributed architecture. The proposed methodology consists of a set of consecutive steps. In the first step they used the Distributed Object-Oriented performance (DOOP) model as a technique for the assessment of relationships between system classes. Next, a recursive graph clustering technique were utilized to partition the Object Oriented system into subsystems that have low coupling and were more suitable for distribution. Then, the mapping process is applied to map the generated partitions to the nodes of the target architecture. An experimental case study was done to illustrate the results after applying each step of the proposed approach.

S.K.Mishra *et al*. [20] Due to the constant change in technology, lack of standardization, difficulties of changes and absence of distributed architecture, the business value of legacy systems have become weaker. We cannot undermine the importance of legacy systems because some of their functions are too valuable to be discarded and too expensive to reproduce. The software industry and researchers have recently paid more attention towards the component based software development to enhance the productivity and accelerate time to market. Instead of re-implementing the business critical applications with-in time and resource constraints, the best option is Software Reengineering (SRE) with effective design and architecture which can make the system better for reusability and maintainability. The main motive behind the reengineering is integrating the legacy system with emerging technologies. To achieve these goals, we have proposed a systematic and concrete model named as Component Oriented Reverse Engineering (CORE). It aims to identify and develop reusable software components. By using the reverse engineering techniques; we can extract architectural information and services from legacy system and later on convert these services into components

Pham Thi Quynh *et al.* [21] Service-oriented systems have become popular and presented many advantages in develop and maintain process. The coupling is the most important attribute of services when they are integrated into a system. In this paper, we propose a suite of metrics to evaluate service's quality according to its ability of coupling. We use the coupling metrics to measure the maintainability, reliability, testability, and reusability of services. Our proposed metrics are operated in run-time which bring more exact results.

Byron J. Williams and Jeffrey C. Carver [22] have described in the proposed work a systematic literature review of software architecture change characteristics. The results of the systematic review were used to create the Software Architecture Change Characterization Scheme (SACCS). The proposed report addressed the key areas involved in making changes to software architecture. SACCS's purpose is to identify the characteristics of a software change that would have an impact on the high-level software architecture.

Sandhu et *al*. [26] have proposed that the Metrics is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. Software metrics were instruments or ways to measuring all the aspect of software product. Those metrics were used throughout a software project to assist in estimation, quality control, productivity assessment, and project control. Object oriented software metrics focus on measurements that were applied to the class and other characteristics. These measurements convey the software engineer to the behavior of the software and how changes can be made that will reduce complexity and improve the continuing capability of the software. Object oriented software metric can be classified in two types static and dynamic. Static metrics were concerned with all the aspects of measuring by static analysis of software and dynamic metrics were concerned with all the measuring aspect of the software at run time. Major work done before, was focusing on static metric. Also some work has been done in the field of dynamic nature of the software measurements. But research in that area was demanding for more work.

3. DISTRIBUTED OBJECT ORIENTED SYSTEMS

A distributed system is one in which components located at networked computers and devices communicate and coordinate their actions by message passing for sharing resources and for distributing computational workload. In a distributed object system, service providers are distributed across a network of systems and are called servers. Clients, who issue requests for those services, are also distributed across a network. A single program can be both a client (issue request) and a server (service request). Clients can issue requests to local and remote servers [8]. A large number of Distributed Object Oriented (DOO) systems have been developed for solving complex problems in various scientific fields. In a distributed object-oriented application, classes can run on a separate computer within a network system. So, they should be distributed efficiently among different nodes. However, the initial design of the DOO application does not necessarily have the best class distribution. In such a case, the DOO software may need to be restructured. The challenge resides in the complexity of interactions between objects [19].

Although these restructuring DOO applications and then mapping the resultant modules to a specific multiprocessor architecture. A distributed OO application consists mainly of a set of interacting objects; each one runs on a separate computer within a network system. There have been a large number of projects related to the use of object-oriented approaches for the design of problem solving environments for complex applications in various scientific fields [7]. The advent and wide adoption of the Java(tm) programming language and environment has provided an opportunity to develop object-oriented techniques on network environment [24]. Designing a DOO application involves an important phase to tune performance by “concretizing” object locations and communication methods [25]. DOO development users can not only develop software products in an object-oriented method but also participate in synchronous/asynchronous collaboration using Object Modeler, Shared Scratchpad and audio communication facility. Large-scale client/server applications distribute objects and then execute across multiple machines. A critical step in performance tuning for these distributed applications
is the identification of hot spots where there is extensive remote method invocation (RMI).

The Dynamic Coupling Measure (DCM) of an object (P) during a time period, denoted by $DCM(P) \mid t$, is defined [28] as the sum over all program execution steps and the sum over the total number of objects, (Oi), coupled to

$$DCM(P) \mid t = \sum_j \sum_i fi(tj) X\ gp(\mid Oi \mid) \qquad (1)$$

where: $i = 0,1,2,.....$ number of coupled objects, $\sum i$ is the sum over the set of objects coupled to P, t is an ordered sequence of program execution steps $< t0,..,tj,..,tn >$ and $\sum j$ is the sum over the pro-gram execution steps; $fi(tj)$ assumes values 1 or 0 depending on whether coupling of the i[th] object is active at tj or not, X denotes normal multiplication and $gp(|Oi|)$ denotes the complexity measure of the i[th] object coupled to. Time units are defined as the individual program steps during program execution. When an object P is created at some time t (i.e., at some step during program execution) [1], the default value of $\mu(P)$ is determined by the (static) class structure.

In the ontological model of [34], a system is a set of things (objects) in which every object is coupled to at least one other object in the set. At the system level, the coupling measure in a time interval t is the sum of all measures defined in formula (I) over all the objects of the system, i.e.,

$$DCM\,(system)\,|\,t = \sum (over\ all\ system\ objects)\mu(k) \qquad (2)$$

The form of $gp(|Oi|)$ depends, as observed by Briand et al. [38], on the goal of the measurement. The goal is determined partly by the external attribute or quality we associate with our measure. DCM is concerned with the internal attribute of object coupling.

4. AN EFFICIENT DYNAMIC COUPLING MEASUREMENTS

Here we propose a hybrid model in Distributed object oriented Software for coupling measurements dynamically. In the proposed method, there are three steps such as Instrumentation process, Post processing and Coupling measurement. Initially the instrumentation process is done. In this process the instrumented JVM that has been modified to trace method calls. During this process, three trace files are created namely .prf, .clp, .svp. In the second step, the information in these file 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 JVMs such that the path of every remote call from the client to the server can be uniquely identified. Finally, the coupling metrics are measured dynamically. The brief explanation of the proposed method is described as follows

4.1 Introspection Process

There are several different techniques for collecting run-time execution information, including techniques such as sampling and direct instrumentation. Sampling requires the running application to be stopped periodically to obtain information on methods that are currently being executed. The accuracy of the information obtained through sampling is determined by the sampling frequency. A higher sampling frequency can provide more detailed information, but this greater detail comes at the expense of greater perturbation of the executing program. Direct instrumentation, on the other hand, adds code to the JVM to directly measure method execution times. This approach provides more precise information than sampling, since no execution steps are missed. However, the additional instrumentation code may produce greater perturbations than sampling. When humans introspect, they look inside themselves. When our programs introspect, they also look inside themselves, but in a different way.

In object-oriented languages, introspection is the idea that program code can get information on itself. In object-oriented languages, introspection permits programs to get and use information on classes and objects at run-time. Using introspection, one can ask an object for its class, ask a class

for its methods and constructors find out the details of those methods and constructors, and tell those methods to execute. Introspection is otherwise known as instrumentation process. The java class files are executed by instrumented JVM to generate the execution events. Introspection allows the main application to examine each of its plug-ins for the methods it supports and then call them when appropriate. Without introspection, it could not determine the names and parameter types of those methods. To minimize perturbation, the JVM was modified only to the extent necessary to generate enough trace information to visualize the execution call graph. Introspection procedure are:-

1. First we have to compile the source code.
2. Then use Reflection to retrieve data members and methods.

Finally maintain a Vector to store the retrieved information

4.2 Trace Events

The trace generation module of the JVM is modified to record every invocation of a method using time stamps that show the start and end times of the method with microsecond resolution. As a Java program is executed by the instrumented JVM, three trace files are generated, i.e., .prf, .clp and .svp files.

.prf file

The .prf file contains detailed trace information that records call and return time stamps for every method executed. Invocations of the same method executed under different threads are distinguished from one another by their unique thread identifiers. The other two files record the client/server interaction, if any, that occurs on the JVM as the program is being executed.

.clp file

The .clp file contains information about all of the outgoing RMI calls from the running JVM, i.e., identifying information for remote methods invoked by this JVM.

.svp file

The .svp file records information about all incoming RMI calls, i.e., all of the methods remotely invoked on this JVM by other JVMs. The .clp file is referred to as the client profile of remote methods for which the JVM acts as a client, and the .svp file is referred to as the server profile of remote methods for which the JVM acts as a server. Note that a server JVM may also execute client- type functions and a client JVM may also act as a server to other JVMs. The trace event algorithm for our proposed work is shown below,

For each $c \in C$
  For each $m \in Mc$
    Let fem be the first line of the method
      $mn = mf \ \& \ mt \ \& \ ml$
  End
End

Fig. 1: Pseudo code for trace events

where, *mf* is Content of the method from beginning to the first line of the method and *ml* is Content of the method from second line to the last line of the method. *mt* is trace event statement to be added. & is the append operator. The above procedure is iterated until all the classes have been processed.

The trace generation module of the JVM is modified to record every invocation of a method using time stamps that show the start and end times of the method with microsecond resolution. Additionally, a thread identifier is recorded to uniquely identify the thread executing the method. The profiling function creates a new trace record in a buffer for each method entry or exit event. For faster processing, the trace information is stored in main memory and written to external files only when the buffer overflows. Since disk operations are time consuming, it is important to minimize the number of rights to the external file. Since the number of methods called within a program can be quite large, and since each instance of a method generates a trace entry, the amount of trace data generated for a large application would be enormous. Filtering options are provided with the instrumented JVM to reduce the amount of trace data collected. The second filtering option specifies a list of classes to be traced. For this option, the class hierarchy of the object on which the current method is invoked is checked to see whether it belongs to any of the classes specified. If so, the method is traced. Otherwise, the method invocation is simply ignored.

Client/server traces generation. The Java remote method invocation (RMI) facility allows one JVM to execute a method on another JVM, which may be executing on a physically distributed processor. To match corresponding entries in the server and client profiles, the modified JVM also records the machine (JVM) names. On the client side, the server machine name on which the call was invoked is recorded. On the server side, the client machine name from which the call originated is also recorded. The client-side port number of the TCP/IP connection used for the remote call is recorded in both the server and the client profiles. The port number is needed to distinguish between remote calls made to the same server from different client JVMs residing on the same physical machine. The identifier of the thread that invoked the remote call is recorded on the client side in order to map the detailed trace entry of the remote method invocation to the corresponding client profile entry. Similarly, the identifier of the thread where the remote call is received is recorded on the server side. Finally, the time stamps the time at which the remote call was invoked on the client and the time at which the call was received on the server are also recorded.

4.3 Post processing

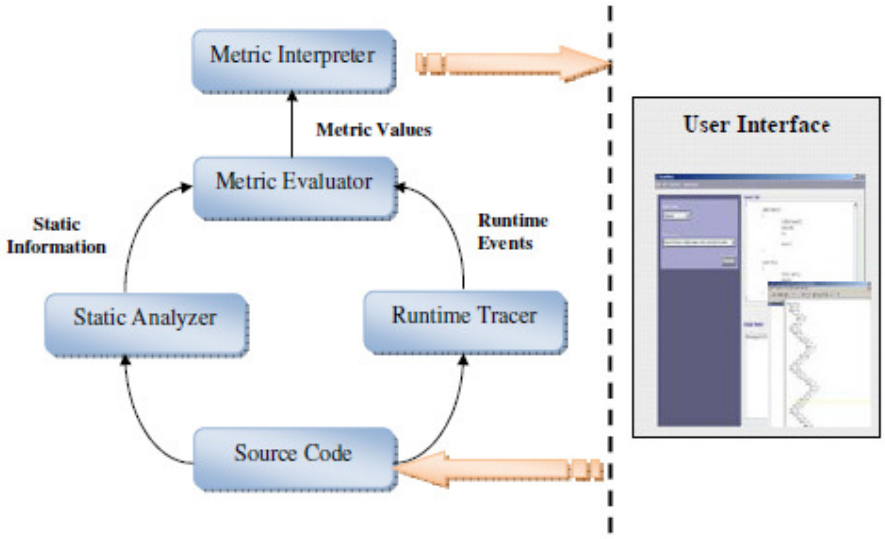
The post processing step takes the detailed .prf profile of each JVM, along with the .clp and the .svp files, as input. The merge and tree generation sub steps process these input files to produce a dynamic execution tree for the desired client or server. The details of these steps are described in the following subsections.

4.4 Merge Step

The main part of the merge process is to link client calls recorded in the client profile of a client JVM to the appropriate entry in the server profile of the remote server JVM where the method was actually executed. To see how the corresponding client- and server-side entries are matched, consider a simple scenario where there is one client JVM interacting with one server JVM. In a single-threaded application, multiple calls made from the client to the same remote object and method on the server can easily be matched using remote object and remote method identifiers recorded in the client- and server-profile entries. The entries with the same remote object and

remote method identifiers on both sides can be aligned in chronological order and the entries matched in that order. Since the JVM is multithreaded, however, there could be a second call made to the server from the client through another thread before the first call even begins, if Java threads are mapped to different native threads in the JVM implementation. Thus, the second call may arrive at the server before the first one. The remote object and remote method identifiers are not sufficient to unambiguously link the appropriate calls in this case. Consequently, the port number of the TCP/IP connection is used to resolve this ambiguity.

Fig.1:



Dynamic Coupling Measurement

4.5 Dynamic Afferent Coupling

Percentage of number of classes accessing the methods of a class at runtime to the total number of classes. It is defined [27] as

$$DCa(C) = \frac{M}{N} \times 100 \qquad (3)$$

Where M is the Number of classes invoking the methods of class C at runtime and N is the Total number of classes. Ca counts the number of classes accessing the methods of a class evaluated from the static code. This count can vary at runtime as there may be some method calls that are counted by Ca but are not executed at runtime. DCa takes account of this problem as it is calculated at runtime only.

4.6. Estimating Dynamic Behavior

Because of the increasing complexity of the systems, it is often hard for developers to clearly understand the behavior of the application. When the system is deployed and running and the performance goals are not met, it is difficult if not impossible to determine where problems are. Tracing a program is used to understand how the program executes and reveals the dynamic details of the design. After Events are added, the code along with the trace events is compiled and

executed. During runtime, the actual function calls Ma are extracted based on the added trace events given by (1).

$$Ma = M \bigcap Mt \qquad (4)$$

Dynamic metrics can be used as improving agents for static metrics by extracting the real time behavior of a software at runtime and taking the appropriate steps (such as eliminating a class that is never used at runtime). After the required changes made in the design phase, value of the static metric can be checked against a threshold value to decide whether there is a further need of dynamic analysis or not [29].

4.7 Dynamic Coupling Measurement

Software metrics are instruments or ways to measuring all the aspect of software product. These metrics are used throughout a software project to assist in estimation, quality control, productivity assessment, and project control. Object oriented software metrics focus on measurements that are applied to the class and other characteristics. These measurements convey the software engineer to the behavior of the software and how changes can be made that will reduce complexity and improve the continuing capability of the software. Object oriented software metric can be classified in two types static and dynamic. Static metrics are concerned with all the aspects of measuring by static analysis of software and dynamic metrics are concerned with all the measuring aspect of the software at run time. Major work done before, was focusing on static metric. Also some work has been done in the field of dynamic nature of the software measurements. But research in this area is demanding for more work.

The metrics available for coupling measurement belong to two major categories i.e. static and dynamic. Static metrics that are applied to the design/source code can only measure the expected coupling behavior of object-oriented software and not the actual behavior. This is because the behavior of a software application is not only influenced by the complexity but also by the operational or runtime environment of the source code [31]. Dynamic metrics on the other hand can capture the actual coupling behavior as they are evaluated from data collected during runtime. A metric should measure an aspect of a program that can only be obtained by actually executing it. The dynamic nature of a metric makes it unaffected by the addition of unexecuted code to the program; because code that is never executed will obviously never contribute to the measured value.

Dynamic metrics demonstrate the actual behavior of software at runtime in comparison to the static metrics that provide an idea of the actual behavior. Dynamic metrics can be defined as the metrics used to measure the internal quality attributes of object-oriented systems by working on the information gathered from the runtime behavioral analysis of such systems. There are just a few dynamic coupling metrics proposed till date [30, 28, 31, 32, 33]. Dynamic coupling measures were introduced as a refinement to existing coupling measures due to gaps in addressing polymorphism, dynamic binding, and the presence of unused code by static structural coupling measures [25]. After estimating the dynamic behavior, we have to filter out the dynamic function calls from the source code by using the eqn. (2). And then the new set of codes with Ma is generated.

5. EXPERIMENTAL 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.

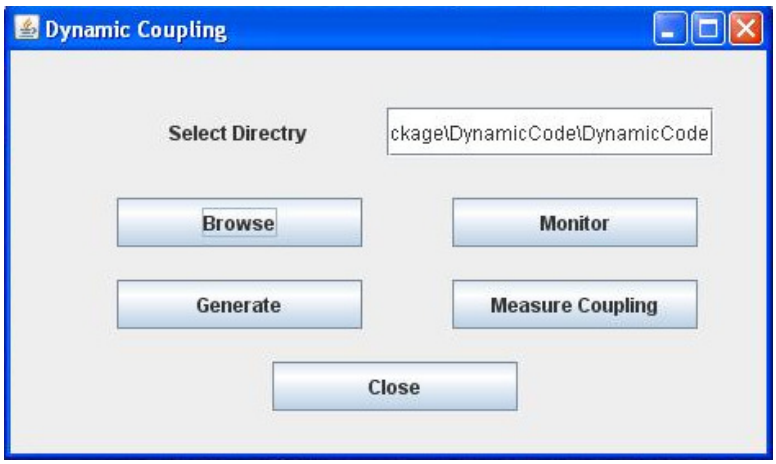


Fig. 2: Initial screen obtained in the proposed system

When we execute the proposed method, the initial screen obtained which is described in Fig. 2. In this, the browse button is used to select the package. In Fig.3, the instrumentation process is described.

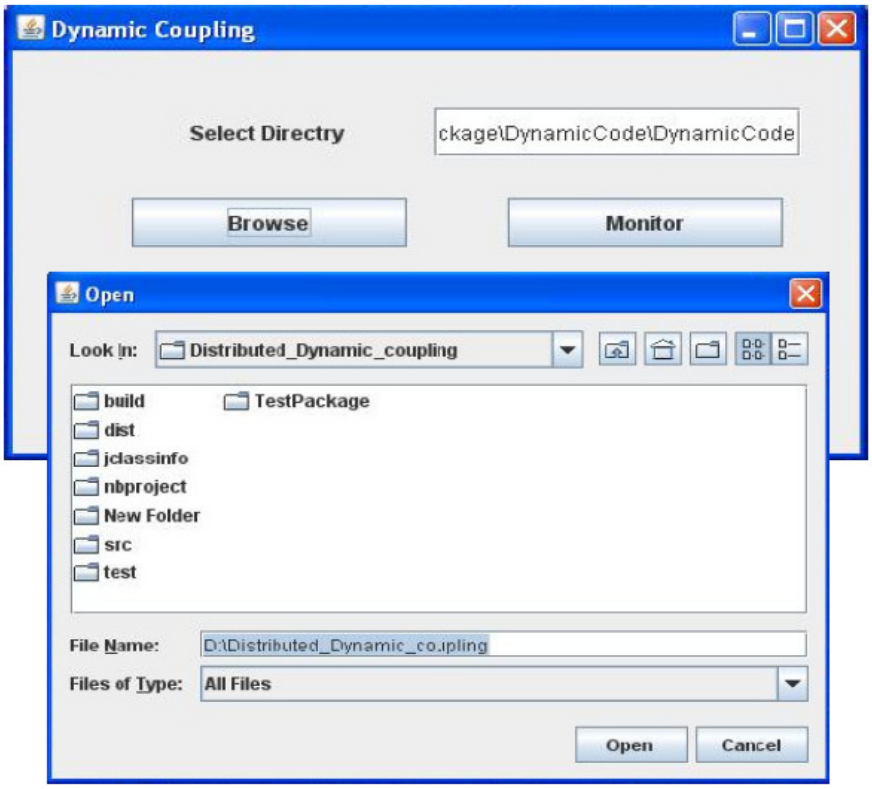


Fig. 3: The sample output obtained in the Instrumentation process.

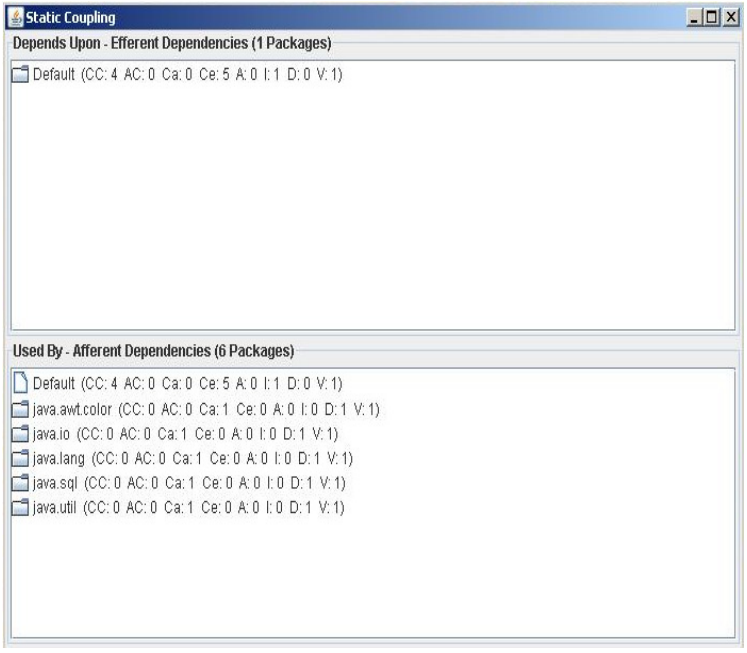


Fig.4: The static coupling measurements

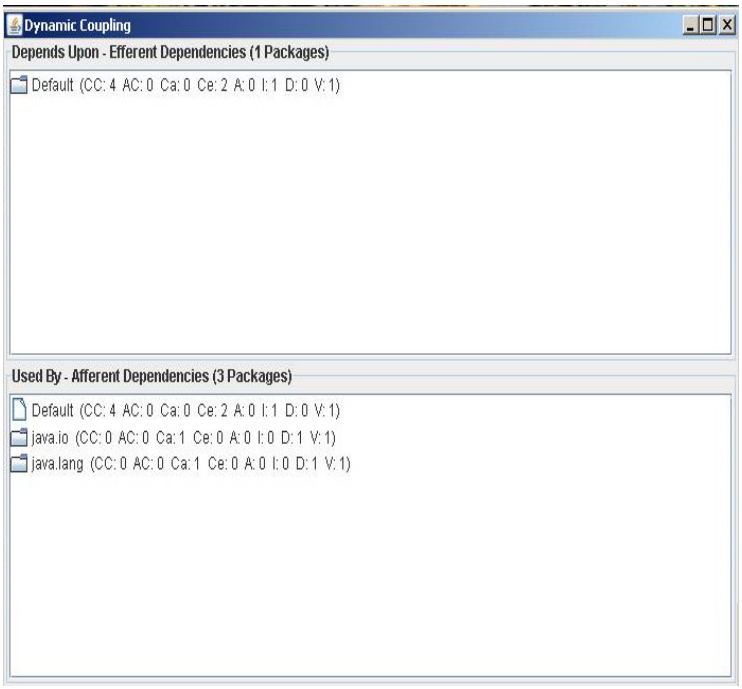


Fig.5: The dynamic coupling measurements

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

6. CONCLUSION

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.

REFERENCES

[1] David Kung, Jerry Gao, Pei Hsia, Yasufumi Toyoshima, Chris Chen, Young-Si Kim, and Young-Kee Song, "Developing an Object-Oriented Software Testing and Maintenance Environment", Communications of the ACM, Vol. 38, No. 10, pp.75-87, oct.1995.

[2] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorenson, "Object-Oriented Modeling and Design", Prentice Hall, 1991.

[3] I .Graham, "Object-Oriented Methods" Addison-Wesley, 1994.

[4] Alan C. Y. Wong, Samuel T chanson, S. C. Cheung and Holger Fuchs, "A Framework for Distributed Object-Oriented Testing", In Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTEX) and Protocol Specification, Testing and Verification (PSTV XVII), pp.1-22, 1997.

[5] L. Lamport and N. Lynch, "Distributed Computing: models and methods", Handbook of theoretical Computer science, pp.1157-1199, Elseiver Science, 1990.

[6] Safwat H. Hamad, Reda A. Ammar, Mohammed E. Khalifa and Ayman El-Dessouky, "A Multi step Approach for Restructuring and Mapping Distributed Object-Oriented Software onto a Multiprocessor System", In proc. of the INFOS2008, March 27-29, 2008 Cairo-Egypt, pp.PAR19-PAR23, 2008.

[7] N. Koziris, M. Romesis, P. Tsanakas and G. Papakonstantinou, "An Efficient Algorithm for the Physical Mapping of Clustered Task Graphs onto Multiprocessor Architectures", In Proceedings of the 8th EuroPDP, pp. 406-413, Jan.2000.

[8] R. S. Chin and S. T. Chanson, "Distributed Object-based programming systems", ACM Computing Surveys, Vol.23, No.1, pp.91-124, 1991.

[9] C. Atkinson, "Object-Oriented Reuse, Concurrency and Distribution: an Ada-based approach", Addisson-Wesley, 1991.

[10] T. W. Ryan, "Distributed object technology: concepts and applications", Prentice Hall, 1997.

[11] Mandeep Kaur, Parul Batra & Akhil Khare, "Static Analysis And Run-Time Coupling Metrics", International Journal of Information Technology and Knowledge Management, Volume 3, No. 2, pp. 707-710, 2010.

[12] Arisholm E., Briand L. C., and Foyen A., "Dynamic coupling measurement for object-oriented software", In proc. of the IEEE Transactions on Software Engineering, vol. 30, no. 8, pp. 491-506, Aug.2004.

[13] Aynur Abdurazik, "Coupling-Based Analysis Of Object-Oriented Software", A Dissertation Submitted to the Graduate Faculty of George Mason University, pp.1-247, 2007.

[14] Yin Liu and Ana Milanova, "Static Analysis for Dynamic Coupling Measures", In Proc. of the 2006 conference of the Center for Advanced Studies on Collaborative research (CASCON '06), pp.1-12, 2006.

[15] A. Kavitha and Dr. A. Shanmugam, "Dynamic Coupling Measurement of Object Oriented Software Using Trace Events", In proc. of the 6th International Symposium of Applied Machine Intelligence and Informatics, 2008 (SAMI 2008), pp. 255-259, Jan. 2008.

[16] Kai Qian, Jigang Liu and Frank Tsui," Decoupling Metrics for Services Composition", In Proc. of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR 2006), pp.44-47,2006.

[17] Benjamin A. Allan, Robert Armstrong, David E. Bernholdt , Felipe Bertr, Kenneth Chiu , Tamara L. Dahlgren , Kostadin Damevski , Wael R. Elwasif , Thomas G. W. Epperly , Madhusudhan Govindaraju , Daniel S. Katz , James A. Kohl , Manoj Krishnan , Gary Kumfert , J. Walter Larson , Sophia Lefantzi , Michael J. Lewis , Allen D. Malony , Lois C. McInnes , Jarek Nieplocha , Boyana Norris , Steven G. Parker , Jaideep Ray , Sameer Shende , Theresa L. Windus and Shujia Zhou, " A Component Architecture for High-Performance Scientific Computing", Intl. J. High-Perf. Computing Appl., Vol.20, No. 2, pp. 163-202, 2006.

[18] Liguo Yu," Understanding component co-evolution with a study on Linux", Journal of Empirical Software Engineering Archive, Vol. 12, No. 2, pp.123-141, Apr. 2007.

[19] Safwat H. Hamad, Reda A. Ammar, Mohammed E. Khalifa and Ayman El-Dessouky,"A Multistep Approach for Restructuring and Mapping Distributed Object-Oriented Software onto a

Multiprocessor System", The 6th International Conference on Informatics and Systems, pp.27-29, Mar. 2008.
[20] S.K.Mishra, D.S.Kushwaha and A.K.Misra, "Creating Reusable Software Component from Object-Oriented Legacy System through Reverse Engineering", Journal of Object Technology, Vol.8, No.5, pp.133-152, Aug.2009.
[21] Pham Thi Quynh and Huynh Quyet Thang ,"Dynamic Coupling Metrics for Service Oriented Software", International Journal of Electrical and Electronics Engineering, Vol. 3, No.5, pp.282-287, 2009.
[22] Byron J. Williams and Jeffrey C. Carver," Characterizing Software Architecture Changes: A Systematic Review", Information and Software Technology, Vol.52, No.1, pp. 31-51, Jan.2010.
[23] Mark C. Campbell, David K. Hinds, Ana V. Kapetanakis, David S. Levin, Stephen J. McFarland, David J. Miller and J. Scott Southworth, "Object-oriented perspective on software system testing in a distributed environment", Hewlett- Packard Journal, Dec. 1995.
[24] Jim Waldo, "Object-Oriented Programming on the Network", In proc. of the 13th European Conference of Object Oriented Programming (ECOOP), LNCS 1628, pp. 441-448, 1999.
[25] Jim Waldo, Geoff Wyant, Ann Wollrath and Sam Kendall, "A Note on Distributed Computing", In proc. of the Technical Report, Sun Microsystems Laboratories, Nov. 1994.
[26] P. S. Sandhu and G. Singh, "Dynamic Metrics for Polymorphism in Object Oriented Systems", Journal of World Academy of Science, Engineering and Technology, Vol. 39, pp. 354-358, 2008.
[27] P. Singh, P. and H. Singh, "DynaMetrics: a runtime metric-based analysis tool for object-oriented software systems", ACM SIGSOFT Software Engineering Notes, Vol. 33, No. 6, pp. 1-6, 2008.
[28] Youssef Hassoun, Roger Johnson and Steve Counsell, "A Dynamic Runtime Coupling Metric for Meta Level Architectures", In Proceedings of Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '04), pp. 339-351, 2004.
[29] Vincent Ribaud and Philippe Saliou, "Towards an Ability Model for Software Engineering Apprenticeship", Italics, Informatics Education Europe, Vol. 6, No. 3, July 2007.
[30] Erik Arisholm, "Dynamic Coupling Measures for Object-Oriented Software", In proceedings of 8th IEEE Symposium on Software Metrics (METRICS'02), pp. 33-42, 2002.
[31] Aine Mitchell and James F. Power, "Toward a definition of run-time object-oriented metrics", In Proceedings of 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2003), 2003.
[32] Sherif M. Yacoub, T. Robinson, and H. H. Ammar, "Dynamic Metrics for Object Oriented Designs", In Proceedings of the 6th International Symposium on Software Metrics, pp.50-58, November 1999.
[33] Andy Zaidman and Serge Demeyer, "Analyzing large event traces with the help of coupling metrics", In Proceedings of the 4th International Workshop on OO Reengineering, Universiteit Antwerpen, June, 2004.
[34] Y. Wand and R. Weber, "An Ontological Model of an Information System", IEEE Transactions on Software Engineering, Vol.16, No.11, 1282-1292, 1990.

Authors Profile

Mr. S.Babu has studied B.Tech. (IT) and M.Tech.(CSE), Research scholar, Anna University of Technology Coimbatore and Asst. Prof, Department of Information Technology, Adhi parasakthi Engineering College, Melmaruvathur, Kanchipuram Distrit, Tamilnadu, India. He is having 6 years of teaching experience in Engineering, and published nearly 4 technical papers in both National and International Conference, 2 Papers in International Journals and Life member of ISTE and IETE

Dr. R.M.S Parvathi has studied B.E. (ECE) and M.E.(CSE) from Govt. College of Technology, Coimbatore, and obtained Ph.D. in Computer Science & Engineering from Bharathiar University Coimbatore, She is having total teaching experience of 20 years both in UG & PG Engineering, Also having 1 year Industrial and 8 years R & D Experience. She published nearly 56 technical papers in both national and International conference and also published technical papers in 4 International & 2 National journals. She is Life member of ISTE and CSI, fellow member of IEEE and guiding students for doctoral program. She is the author of 2 books on UML.