Slicing Layered Architecture for Characterizing Reuse-Driven Software Engineering

Divanshi Priyadarshni Wangoo
Assistant Professor
Department of Computer Science Engineering & Information Technology
Amity School of Engineering & Technology
Amity University, Gurgaon, Haryana, India

Abstract— This paper presents an efficient methodology for representing the layered architecture modules in terms of slicing object components. The goal is to reduce the software dependencies in the overall architecture style characterizing the objective of Reuse-driven Software Engineering. The dependency graphs for the high level component systems structures the application systems for making reuse cost-effective and ensures deployment of high quality software systems to the end users. Moreover, the Lay_Slice Dependency Graph helps in aligning the reuse levels in the component systems to a comprehensive pattern that improves the productivity and quality of the reuse business process. The slicing layered architecture ensures a well-defined architecture system with new slicing based Object-Oriented modeling techniques that helps in more systematic dealing with the complexity of large software systems.

Index Terms— Reuse Driven Software Engineering (RESB), Lay_Slice Dependency Graph (L_SDG), Lines of Code (LOC), Unified Modeling Language (UML), Slicing, Application Systems, Component Systems.

1 INTRODUCTION

THIS paper presents an efficient technique for enhancing software reuse in Reuse Driven Software Engineering (RESB). Information systems built on large-scale in today's modern world is very complex and are subjected to standards that are constantly changing. A good software architecture is necessary for managing the intrinsic complexity prevalent in the changing software needs. The architecture of a software system defines that particular system in components that are computationally driven with connections existing among those components. In Object-Oriented systems, the overall software architecture is determined by statically organizing the software into subsystems with interfaces interconnecting the subsystems together. Communication overhead in large organizations increases as coordinating with the organizations distributed geographically requires the software engineers to put more efforts in proposing a good software architecture. A right and well-defined software architecture will reduce much of the above stated problems as it will help the software engineers working on the architecture to better understand the scenario.

Reuse Driven Software Engineering (RESB) or Reuse Business on the organizational level consists of several elements like architectural group, component developers, project groups, support groups and reuse manager all interacting together for organization's successful reuse. For architecting the components and applications in a layered architecture system, the reuse business rests in the three layers of components that go into the application system. Layered modular architectures in which application systems are build form component systems require extensive RESB techniques for making the reuse cost-effective. There is a need of good object-oriented techniques which will make the reuse process more efficient in terms of increased performance, cost and reduction in time to market and propagation of bugs from the reused software. All the reuse engineering efforts are best utilized when all the factors leading to the building of a good architecture system is properly utilized and managed. The new software systems build from the old software systems incorporate all the features existing in the latter to give rise to a new software domain keeping the characteristics of the latter. This capability of reuse systems comes along with inheriting all the features of the reused system including the bugs present in the reused code. The novel systems need to build a reuse mechanism wherein the software inheriting capability comes along with identifying the reused software with promising bugs present so that they are not propagated in the new software system build. This ensures making the reuse business process more systematic and bug-free and thus enhances the cost-effective and quality ensuring approach to development of application systems.

Thus, slicing techniques used in the layered architecture enhances reuse in the application from component systems and supports reuse business processes using various Object-

Oriented slicing techniques. The slicing technique introduced in this text incorporates all Object-Oriented features for dynamically slicing the components in the component system layer of the software layered architecture system.

The objective of the paper is to introduce slicing techniques at the component level of the layered architecture that will make reuse business more effective in terms of cost and time. By reducing the software dependencies at the component level, the application systems will incorporate the reused component which will be free from bugs and thus will enhance the performance of the new system built. Reusing the components in the component systems for building new application in the application systems incorporates incremental and iterative software engineering lifecycle at all stages of the process. As a result, the application systems will be build more efficiently with a key enhancement in delivery of the software in the market with time and cost-effective performances of the software application built.

2 REUSE-DRIVEN SOFTWARE ENGINEERING (RESB)

The Reuse-Driven Software Engineering is composed of four dimensions with an interlocking dimension. The four domains are- business orientation, engineering orientation, technical sequence and business process reengineering [2]. The reuse business rests in the three layers of the components that are inherited by the applications residing on the top layer of the layered system architecture. The goal of reuse business is to produce a novice application system by reusing the components of the defined component systems. This ensures code reusability, saves developers time in writing the same code again, improves quality and accelerates the speed of software delivery to the end user. Thus, managing the reuse business is the most critical part for and overall development of a reuse business organization.

2.1 Incremental Systematic Reuse for Concurrent Processes

The Software Development Lifecycle process encompasses all the stages necessary for building a software application form planning, communication, analysis, design to implementation, testing and deployment to the end user [1]. Software engineering practices are rigorously scrutinized so that the resultant software is produced with enhanced performance and productivity. Most of the software systems are build incrementally. The step by step development of a system is usually released in a series of versions, releases, increments or cycles. The various models are iterated with each increment and modeling activities are overlapped with the identification of associations among the models. Efficient debugging processes require a lot of effort. Debugging process aims at finding, localizing, modifying and correcting errors by setting breakpoints [4]. If bugs are identified at an early stage of software development much effort of the developer is utilized in making the software more efficient and cost effective.

Thus, building of a good system architecture requires more careful analysis and design processes along with implementation of all the important steps described for the software development lifecycle.

3 SOFTWARE LAYERED SYSTEM ARCHITECTURE

A layered system architecture is a software architecture that organizes software in layers and each layer is constructed on the top of more general layer [2]. A layered system encompasses system interaction all at levels and in both horizontal and vertical dimensions. Analyzing the static dependencies among the systems in the layered architecture leads to identification of associations from some generalized systems to more specific systems in a bottom-up hierarchy. The vertical dimension of the layered system helps in identification of static dependencies existing among the systems across the layers. The dependent component systems can be extracted for effective utilization into the application systems that would result in more efficient and optimized software production.

3.1 Modeling Layered Architecture with Slicing Methodology

The application of slicing methodology to the layered system architecture is important from the reusability of component systems viewpoint. As the old component systems are analyzed for their productive capacity in building of a new software systems, some important factors such as bugs remain hidden from the developer leading to more efforts in removing the bugs first from the component system and then from the bugs inherited in the application systems. The system prototype of the building software is needed for rapid iterative development resulting in controlled costs and early availability of the prototype for experimentation purposes [5]. Slicing methodology encompasses the use of various dependence graphs involving all the object-oriented features such as objects, classes, encapsulation, abstraction, inheritance through promoting reusability. The slicing criterion for program slicing is represented by <*v*, *p*>, where *v* is the program variable and *p* is the program point [3]. Thus, slicing enhances all the object-oriented features for promoting efficient reusability of software components and helps in managing the reuse business process.

4 IMPLEMENTING LAY_SLICE IN ARCHITECTURE COMPONENTS SYSTEMS

The layered architecture system is composed of application systems residing at the top level that are built from component systems in lower layers [2]. A layered architecture is a well-defined software architecture that systematically manages as well as organizes software in layers. Each layer in the layered architecture resides at the top level component of a more generalized layer. The top level or upper layers are more specific while the bottom level or lower layers are more general in nature encompassing the type of systems build on the respective layers. The slicing methodology is introduced in the components that reside in the business-specific and middleware layer of the layered architecture. The four layers of the layered architecture have distinct characteristics that dynamically characterize the role of each layer. The topmost layer is the application system layer which contains one application system for each software system, the next layer being the business-specific layer contains a number of business-specific component systems, the middleware layer offers component systems that provides utility classes and platform- independent services and the last layer being the system software layer contains the software for the computing and networking infrastructure [2]. The layered system consisting of application and component systems is described below in Fig. 1 as follows.

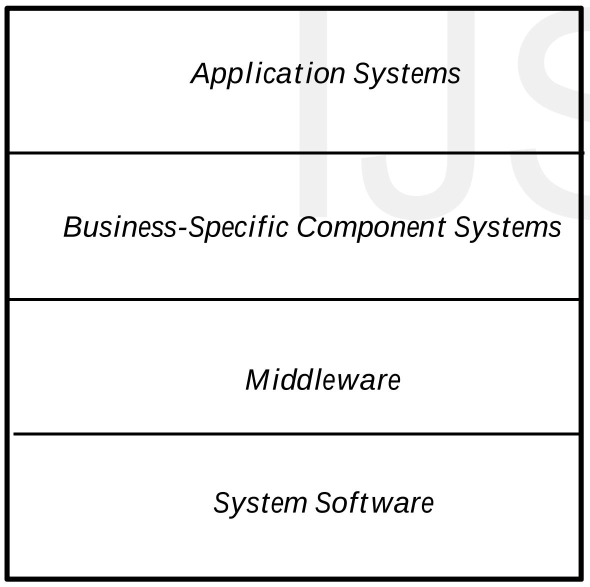

Fig. 1. A layered system architecture consisting of four layers- application, business-specific, middleware and system software

4.1 *LAY_SLICE DEPENDENCY GRAPH (L_SDG)*: ASSESSING LAYERED DEPENDENCY GRAPHS FOR ARCHITECTURE & COMPONENTS SYSTEM MODELS

The LAY_SLICE DEPENDENCY GRAPH (L_SDG) is a layered system dependence graph that takes as input the component system, components to be reused in the component system and the facade name and gives as output the dependencies of the reusable components in the application systems build from the component systems. The dependencies are identified in the lines of code accompanying the system architecture for the software system. The L_SDG computes the dependencies subjected to the following layered slicing criterion

$$D= < CS, c_1 \ldots\ldots c_n, Fa>$$

Where D is the result of the computed dependencies, *CS* refers to the component system consisting of components available for reuse, $c_1 \ldots\ldots c_n$ are individual components in the component system and *Fa* refers to the facades used for importing the reusable components to be reused in the application systems.

The software system of the layered architecture employing reuse business is systematically analyzed with the slicing algorithm *Slice_La* which identifies the dependencies among the components in the component systems of the layered architecture. The components are specifically subjected to the layered slicing criterion which results in the identification of dependent components in the layered system. Thus, the components to be reused in the application systems are recognized and any bug prone or least utilized component can be prevented to be utilized as a reusable component in the application system defined at the application level of the layered system. The *Slice_La* ensures optimum utilization of memory with optimized reduced levels of costs and delivery time to the reuse organization for further enhancing application system product delivery to the end users.

4.1.1. *Slice_La*: Slicing Algorithm for Component Dependencies in Layered Architecture

The *Slice_La* algorithm is a layered slicing algorithm for identifying the dependent components in the component systems to be reused in the development of new application systems which are overall characterized by the layered system architecture. The algorithm takes as input the *L_SDG* and the layered slicing criterion and gives as output the dependent components in the component systems. The algorithm consists of the following steps: -

Slice_La Algorithm:

Step 1: - Start and Construct the *L_SDG* for the specific application system to build based on the layered system architecture.

Step 2: - Specify the individual components $c_1 \dots\dots c_n$ to be reused in the component system *CS* defined for the layered architecture.

Step 3: - Build the facades *Fa* for packaging the components identified for reuse in the application systems

Step 4: - Compute the dependencies D with the help of the layered slicing criterion

$$D = < CS, c_1 \dots\dots c_n, Fa >$$

Step 5: - Locate the identified dependent components which are not compatible for reuse and mark them separately.

Step 6: - Repeat steps 2-4 for identification of new component systems

Step 7: - Reuse the best components in the component Systems for building application systems in the layered architecture.

Step 8: - Stop repeat the algorithm for other layered system architectures.

4.1.2. Reusable components for Design and Implementation

The analysis model is developed by mapping the use case model and by reusing analysis components. The design model is the blueprint of the software code built for the underlying software system. The determination of reusable components is an essential step for designing of the new application systems. Time critical systems that require the delivery of software system in a short span of time requires extensive construction of the application system at the developer's end. The component based software application development resolves most of the developer's problems in managing the reusable components. But bugs present in the existing software made available for reuse persists in the new application as well. In order to identify the components with bugs at an early stage of reuse business process will enhance the work of an application developer along with making the delivery of a bug free software in a cost effective and timely manner.

The ATM transaction process example is used here for identifying the reusable components in the component systems. All the elements in the example are drawn from the point of view of Unified Modeling Language (UML) notations. The UML diagram for the component system MyAccount Transaction is drawn as defined in Fig.2 below: -

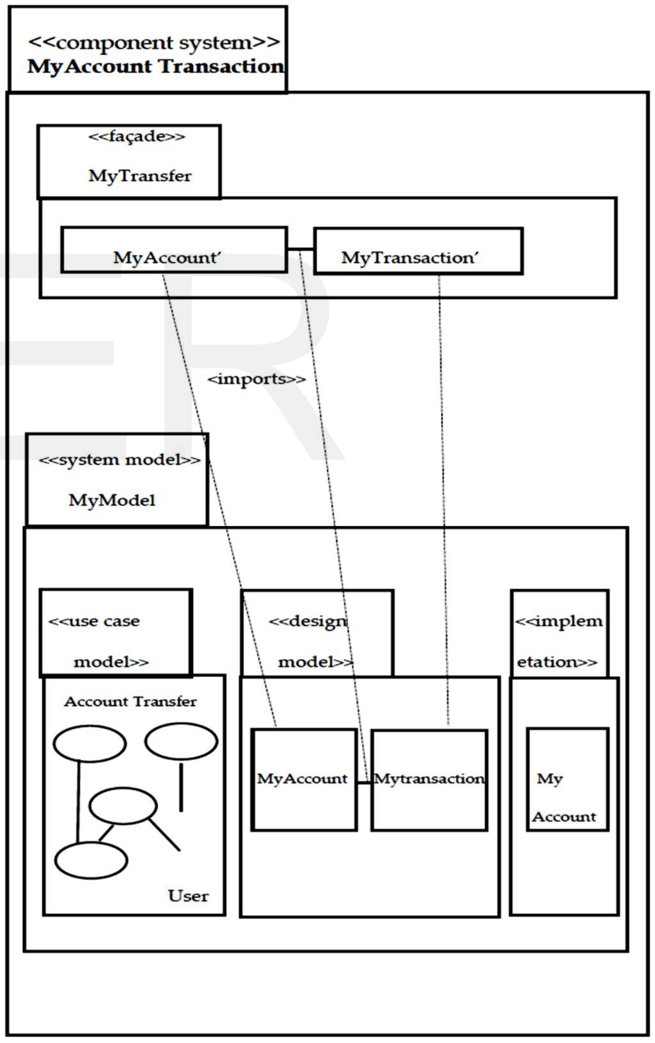

Fig. 2. The Component System MyAccount Transaction

The lines of code (LOC) are defined in Fig.3 and the corresponding L_SDG of Fig.3 is drawn in Fig.4 as below.

```
1.  public class MyAccount Transaction
2.  {
3.  public class systemmodel
4.  {
5.  public class usecasemodel
6.  {
7.  int ac_ta;
8.  }
9.  public class designmodel
10. {
11. public static myaccount (int a)
12. {
13. System.out.println(“the account number is” +a);
14. }
15. public static mytransaction (int ta)
16. {
17. System.out.println(“the transaction number for the account is” +
    ta)
18. }}
19. public class implementationmodel
20. {
21. int ma;
22. }
23. }}
24. public class facademytransfer extends designmodel
25. {
26. public static void main (String args [])
27. {
28. public static myaccount’ (int fa)
29. {
30. System.out.println(“the account number is” +fa);
31. }
32. public static mytransaction’ (int fta)
33. {
34. System.out.println(“the transaction number for the
    account is” + fta)
35. }
36. }
37. }
```

Fig. 3 LOC for the Component System MyAccount Transaction

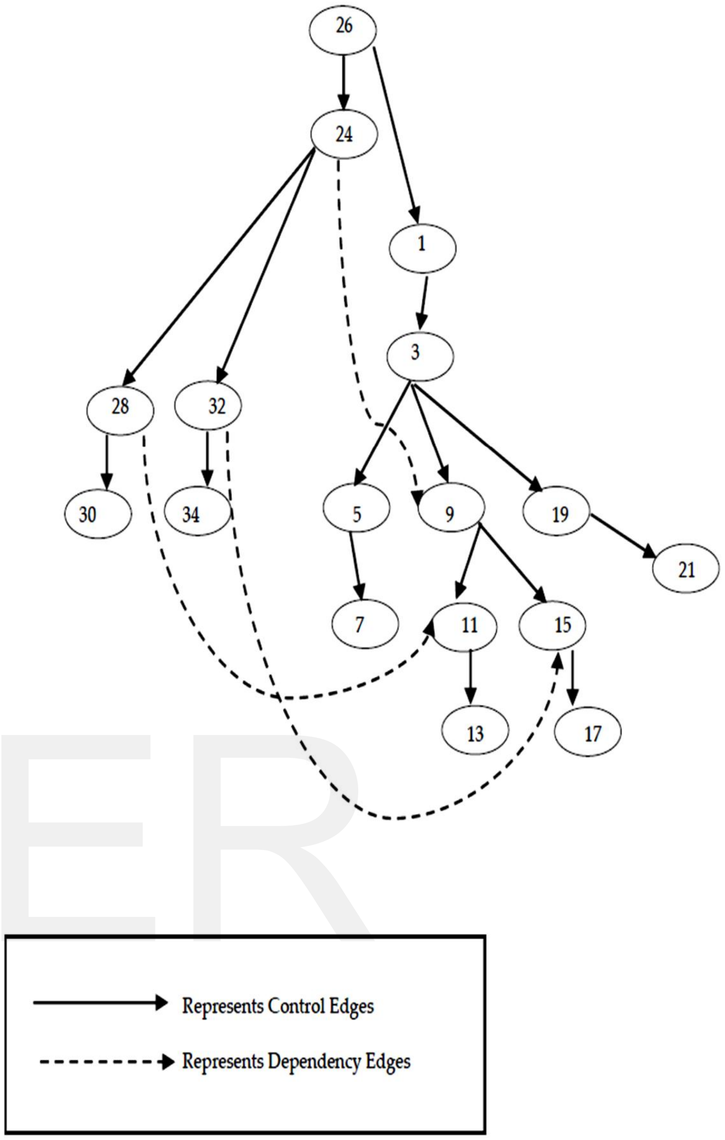

Fig. 4. The corresponding L_SDG for Fig. 3.

The Lines of Code (LOC) defined above represent the reuse criterion of importing components packaged into the facades and reusing the packaged facades in the application system deigned for the layered system architecture. The corresponding L_SDG for the above figure is described next.

. In the above figure, the dependency edges are shown with dashed arrows and control edges with straight arrows. The dependency edges represent the relationships in the UML component diagram where components packaged in the form of facades for reuse purpose.

The complete architecture showing the layered system with the component system and the corresponding application system importing from the component system is shown in Fig.5 as below.

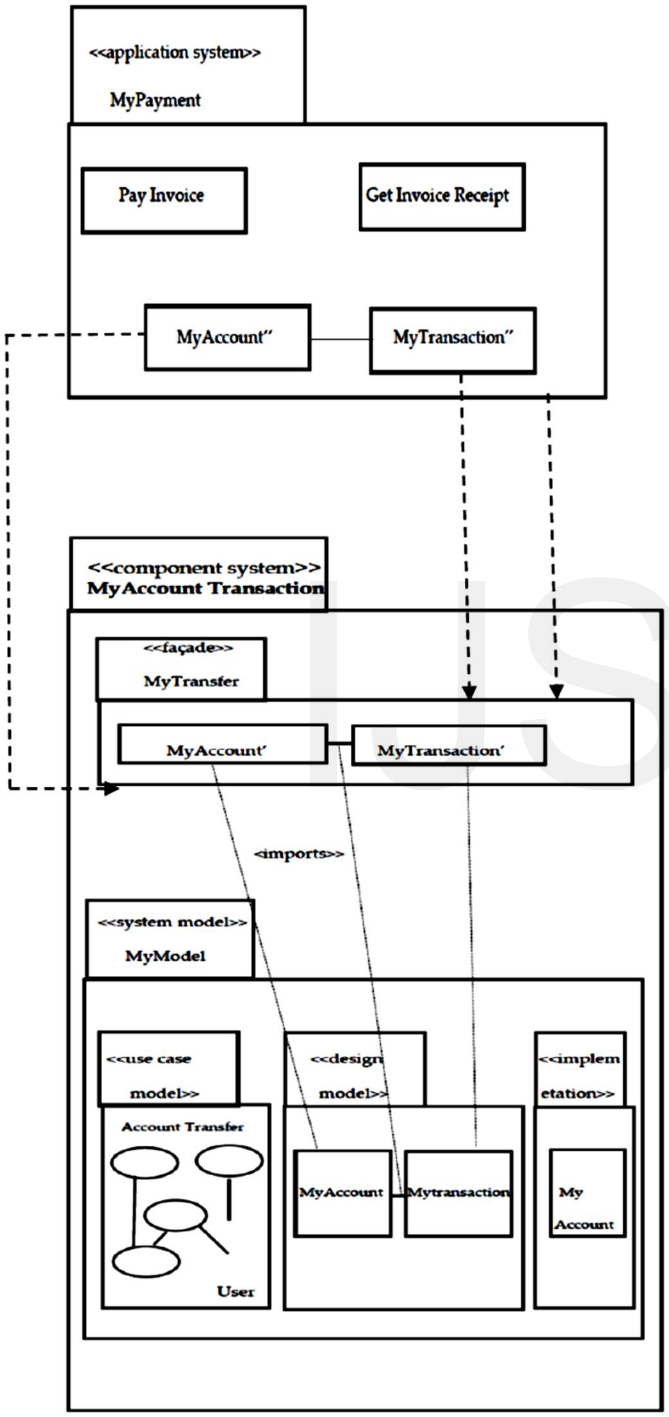

Fig. 5. The Application System My Payment importing components from the Component System MyAccount Transaction

5 *Slice_La* Algorithm Implementation Results

The implementation of *Slice_La* algorithm gives the precise identification and computation of the dependent components either from the components imported in the facades or the facade components imported into the application system for reuse in the layered architecture. The *Slice_La* algorithm produces the following results that are tabulated in Table 1 defined below: -

Table1
Execution results of *Slice_La Algorithm*

S.No	Components in the <<Component System>> $c_1 \ldots\ldots c_n$	<<Application Systems>> in the layered architecture $A_1 \ldots\ldots A_n$ Or <<façade>>	Layered Slicing Criterion *< CS,* $c_1 \ldots\ldots c_n$, *Fa>*	Dependent Components *D*
1.	MyAccount'	<<façade>> MyTransfer	<<MyAccount Transaction, MyAccount', MyTransfer	MyAccount
2.	MyTransaction'	<<façade>> MyTransfer	<<MyAccount Transaction, MyTransaction', MyTransfer	MyTransaction
3.	MyAccount''	<<application system>> MyPayment	<<MyAccount Transaction, MyAccount'', MyPayment>>	MyAccount'
4.	MyTransaction''	<<application system>> MyPayment	<<MyAccount Transaction, MyTransaction'', MyPayment>>	MyTransaction'

6 CONCLUSION

The paper presents an efficient layered slicing algorithm for identifying the concrete components in the component system for reusability in the application systems of the layered architecture. The software dependencies in the layered system are identified in all the derived components and are analyzed prudently for effective importing of the components into the application systems. This ensures managing the reuse business process in the most efficient and cost effective manner along with optimized performance of the new application system build.

REFERENCES

[1] Pressman, S. Roger, “*Software Engineering: A Practitioner’s Approach”*, McGraw Hill, International edition, 6/e,2005.

[2] Ivar Jacobson, Martin Griss, Patrik Jonson, “Software Reuse Architecture, Process and Organization for Business Success”, ACM Press, 2000.

[3] Donglin Liang and Mary Jean Harrold , *“Slicing Objects using System Dependence Graphs,”*International Conference on Software Maintenance, Washington, D.C, pp.358-67, November 1998.

[4] Baowen Xu Zhenqiang Chen, Dynamic Slicing Object-Oriented Programs for Debugging, Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM’02), pp.115-122, 2002.

[5] Ian Sommerville, Software Engineering, Pearson, 9th Edition, 2011.