FP- International Journal of Computer Science Research (IJCSR)
Volume 2, Issue 2, Pages 104-108, June 2015

Compatibility Testing of Component Based Software Modules using Boltzmann Learning

Parul Rajpal
Department of Computer Science
Chandigarh University
Gharaun, India
parul.rajpal1306@gmail.com

Er. Amanpreet Kaur
Department of Computer Science
Chandigarh University
Gharaun, India
raniaman@gmail.com

ABSTRACT
Component-Based Approach is used to revolutionize the development and maintenance of Software systems. In this the distributed system approach is used. Many companies today claim to be doing component based development. All this development comes under the distributed system. In this paper we use Component based software engineering to check the compatibility of the different types of software and it helps to check how much this software is compatible with other software using Boltzmann learning. In the previous work knowledge based learning has been used which is time consuming process. So a novel technique has been proposed in this paper to overcome time constraints.

General Terms
Boltzmann learning Algorithm, Knowledge based learning, Compatibility testing.

Keywords
Software Engineering, Component Based Software Engineering, Distributed systems, Object-Oriented approach.

1. INTRODUCTION

Software is the program or set of programs. It is different from the program in many ways. As in software many things are included: as it consists of the programs, the complete documentation of that program, the procedure that is use to set up the software and the various operation of the software system [8]. Any program is the subset of the software. As on the other hand, program is the combination of source code and object code. There are three major approaches in the software engineering. These are structured, object oriented and component based.

1.1 Structured Approach
This approach contains the basic steps of software development process. This process contains analysis, design, implementation, testing, and maintenance. In this approach some elements are used [2]. The elements are data dictionary, data flow diagrams, state transition diagrams, entity relationship diagrams, process specifications, control specifications, and data object descriptions for analysis model.

Data flow diagrams

These are used for the transformations of data. In this data flows through a system. A data flow diagram consists of processes, data flows, actors, and data stores.

1.1.1 Data dictionary
It contains the missing details from data flow diagrams. In the data dictionary the data flows and data stores in the system.

1.1.2 State transition diagrams
These diagrams show time dependent behavior.

1.1.3 Entity relationship diagrams
These diagrams highlight the relationships between the various kinds of data stores.

1.2 Object Oriented Approach
It is the designing part which further classifying into four parts .The object orientation provides a mechanism to support the reuse of program code, design, and analysis models. In this approach classes and objects are used. UML stands for unified modeling language. UML is the standard language. It is used to visualize, construct, specify, and document the various characters of the project. UML classifies the Structural things into seven parts viz. class, interface, collaborations, use case, active class, components, and node.

1.3 Component Based Approach
Component-Based Approach is used to revolutionize the development and maintenance of Software systems. In this the distributed system approach is used. Many companies today claim to be doing component based development. All this development comes under the distributed system. [1]

The object-oriented approach is extended by distributed approach due to which it extends the ability of calling objects across address space boundaries. The figure 1 below describes the transfiguration that takes place after object oriented approach.

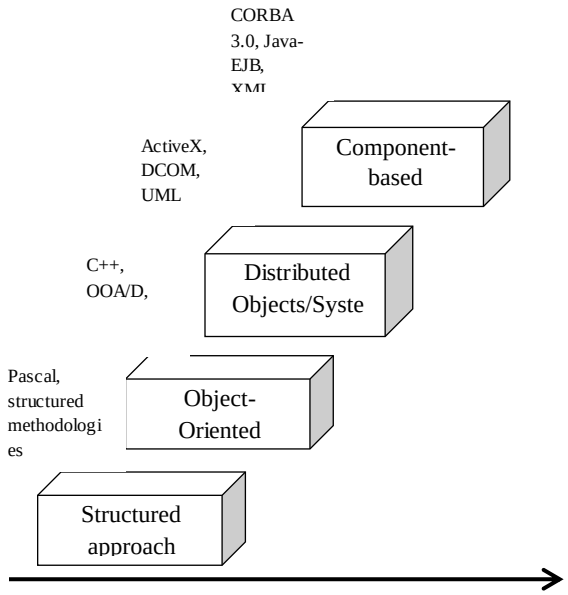

Figure 1 Evolution of Components in the industry [1]

2. LITERATURE SURVEY

In this paper [3] Kuljit Kaur Chahal et.al, has done a survey on current component-based software development which makes use of existing software components to create new applications. Software components may be available within the company or purchased on the global market. One of the most critical in this process is to reuse the basis of selection of the appropriate components. Evaluation component is the core component selection process. Component quality models have been proposed to decide upon a criterion against which candidate components can be evaluated and then compared. But none is complete enough to carry out the assessment. It says users of the components are not necessary to worry about the internal details of the components. However, we believe that the complexity of the internal structure of the component can help estimating the effort related to evolution of the component. In this research, focus is on the quality of internal design of a software component and its relationship to the external quality attributes of the component.

In this paper [9] Prasanta Bose, emphasized that the available evidence in a legacy software system, which can help in understanding and architecture recovery are not always enough. Too often, system documentation is poor and outdated. One could argue that the most reliable information is the source code of the system. However, a significant knowledge about the problem domain is required to facilitate the extraction of useful information from the system architecture. Thus, by providing a mapping between the source and characteristics, the system model and function supports architectural recovery program understanding. The first approach was developed as part of a migration methodology towards a component-based architecture of legacy systems. Retrieving information about features and architecture is collected in a tank to allow passage refactoring as follows. The approach is currently applied on a large project for reengineering of a system of industrial image processing.

In paper [4] Sanjay Misra, has discussed about the use of software metrics. Software metrics can be used to improve the quality and productivity of the system. For designing the object oriented code, complexity factor can be used. In the object oriented system inheritance is used to check the complexity of the system. The design of the system is difficult and expensive to change. The object oriented software development is expensive due to some features like encapsulation, inheritance, polymorphism and reusability. Cognitive designs give the information about the design of object oriented systems. High cognitive complexity leads to the poor design. It is difficult to manage.

In paper [7] P.G Chaitanya, Dr.K.V.Ramesh, stated that the complexity of software in safety critical systems has increased significantly over the last ten years, so that the way of dealing with the complexity and gain high reliable software plays an important role in ensuring the overall product quality.

Leonardo Mariani, Sofia Papagiannakis, this [10] paper emphasized on the COTS component. Software engineers frequently update COTS components. These components are integrated in the component based system. While using the COTS, the two main problems occur. Two tackle these problems, some schemes are proposed in this paper. The problem of quickly identifying components is resolved. In this case the components are compatible with the interface specifications, but badly integrate in target systems. The other problem is regression that is generated in test cases COTS components are third party software modules. These modules are integrated into software products to reduce development time and effort. COTS components are provided as binary code with natural language specifications of integration and deployment procedures.

3. COMPONENT BASED SOFTWAREENGINEERING

Component-based software engineering is the branch of software engineering that emphasizes the separation of concerns. These separations are concern with the wide ranging functionality of the software system. Component based software engineering assembles the software products from preexisting smaller products. These products are known as the components. A component model generally defines a concept of components and rules for their design time composition and is usually accompanied by one or more component technologies, implementing support for composition and interoperation [5]. This practice aims to bring about

an equally wide ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software. Software engineers regard components as part of the starting platform for service orientation. Components play this role, for example, in Web services, and more recently, in service-oriented architectures (SOA), whereby a component is converted by the Web service into a service and subsequently inherits further characteristics beyond that of an ordinary component.

3.1 Software Component

Within the field of software architecture there is a widely accepted terminology where the constituent parts of a system's architecture are, in general, called components. A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard [6]. According to this definition, all components must conform to a component model, which the authors define as specifying interaction and composition standards. This requirement is quite reasonable, since it is hard to see how CBSE could work without some standards for interaction and composition. It is worth noting that the definition does not require that the component model is defined by a standards body or platform supplier, or that a commercial platform implementation is used.

3.2 Characteristics of Components

Following are the characteristic features of a component that distinguishes it:
A component should be: [6]

Standardized: By standardization of a component, we mean that a component being used in a CBSE process should conform to a component model that is standardized. And the model to which it conforms may define component interfaces, meta-data, documentation, composition and deployment.

Independent: In general sense independent means free from outside control or not depending on others for own existence. So, for a component to be independent – the component should be easily composed and deployed without depending on other components. In some circumstances the component may need services from external source, here we need to mention explicitly in a "requires" interface specification.

Composable: the component interacts externally where interaction should take place with the help of interfaces that are defined publicly. This is what makes a component composable. In addition to this, the component should allow external systems to access the methods and attributes and other information about the component.

Deployable: for the deployable property, self-contained and stand-alone entity, are the two keywords essential for a component. Component can be said as a binary which means we need not to compile it before deploying it.

Documented: with all the features aside, it is necessary to have the full documentation of components which makes it easy for the users to decide whether their needs can be fulfilled or not. We need to specify the syntax and semantics of all the interfaces of a component.

Reusable: reusable means ability to be used again and again by other systems or programs. So this characteristic of a component says that it should be designed and implemented in such a way that other program or systems can reuse it again and again.

3.3 Benefits of CBSE

The advent of component software is one of the most important new developments in the software industry since the introduction of high-level programming languages. Component software combines the advantages of custom software and standard software. It enables solutions that are better evolvable and more readily maintainable, can be extended over time and can be modernized incrementally.

3.3.1 Business Benefits

Short time to market: applications with components can be delivered synchronously with the business change cycle time.

Adaptable: componentized applications can be able to respond to change without forcing costly rewrites of the entire system.

Supporting integration: there is often no time or justification to replace legacy applications. New functionality can be integrated with new packages, existing applications and data sources.

Applicable: componentized applications can align with the business. It is unacceptable to require the business to change to the application functionality.

Upgradeable: improvements to an existing function can be possible without impact to other functions.

3.3.2 Technical Benefits

In the view of component users, the technical benefits can be summarized as following:
Power users: will find it second nature to assemble their own personalized applications using off-the-shell components.

Small developers: will find that components reduce expenses and lower the barriers to entry in the software market.

Large developers and System Integrators: will use component suites to create enterprise-wide client/server applications in record time.

Desktop vendors: will use components to assemble applications that target specific markets.

4. PROPOSED METHODOLOGY

The existing software systems face the problem of compatibility. The compatibility is the big issue in the CBSE. Because when software is developed, its components should match. Their compatibility effects, when the environment of the components that we use in the software is totally different. If the software developers select the components manually, they are unable to check the functionalities of these components. It also leads to a problem called compatibility problem. In this work, technique has been proposed to analyze the compatibility of the component based software modules. The component based software modules are plug and play based, means independent modules. In this work, dependencies have been calculated between various component based modules to analyze the compatibility. To analyze the compatibility neural network technique has been applied. The dependency between the modules will be analyzed on the basis of number of values transferred from one module to other module. To analyze the compatibility dependency graphs has been drawn. To draw dependency graph, input training dataset values has been considered. The training dataset values are initial assumption that how many values are exchanged between modules. To calculate dependency at each iteration, initial value will be incremented by 1 and check the error if the error will reduce further increment is done. In the project one stage will come at which error become stable or keep on increasing will every increment, now that value will be considered as the final dependency value of two modules.

In figure 2, software is in idle state. Here four software boxes are taken. These component boxes contain the various components.

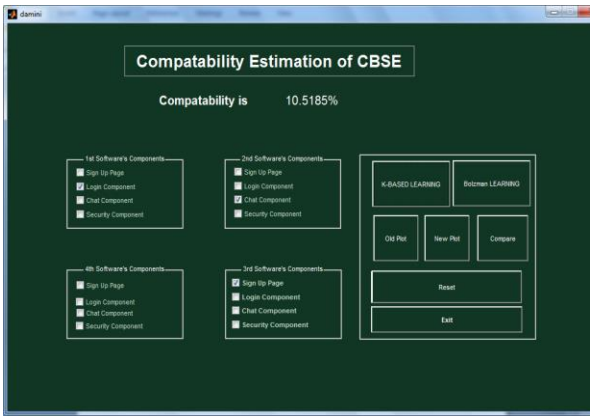

Figure 2 Ideal State

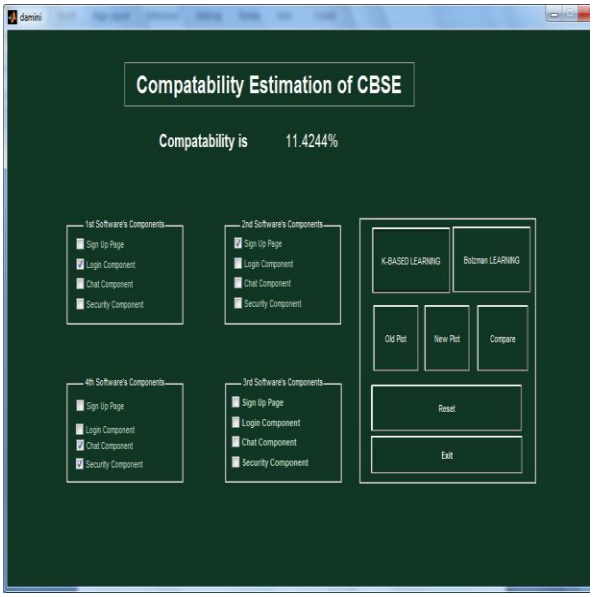

Figure 3 Compatibility Check using Knowledge Based Learning

In figure 3, the compatibility between the various components is shown using knowledge based learning technique.

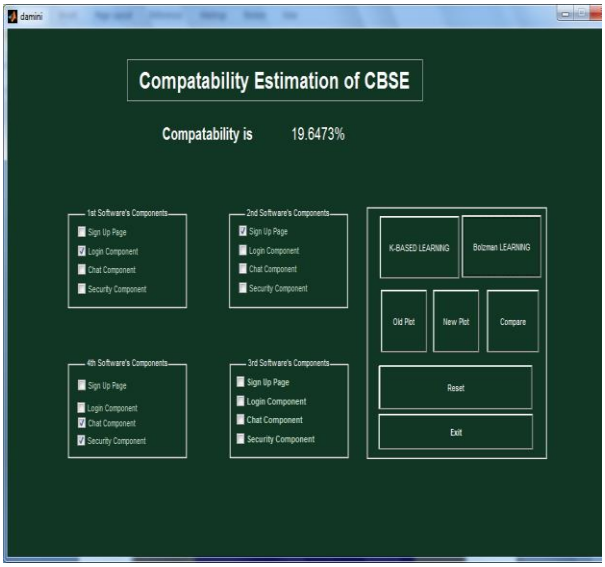

Figure 4 Compatibility Check using Boltzmann Learning

In figure 4, the compatibility between the various components is shown. The computability between various components are shown using Boltzmann learning.

As illustrated in figure 5, the techniques of knowledge based learning and Boltzmann learning techniques are applied and in figure compatibility of the two techniques are compared graphically.

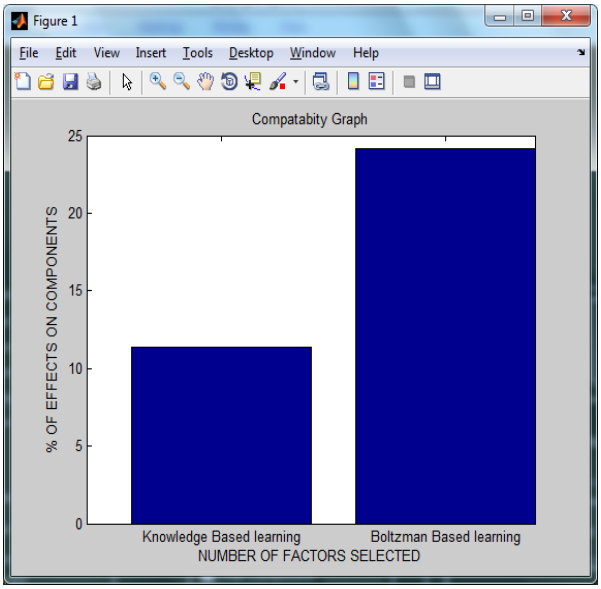

Figure 5 Comparison Graph

5. CONCLUSION

The component based software modules are integrated when required and the basis of compatibility. To analyze the compatibility of the component based software modules technique of neural network had been proposed. In the previous times, knowledge based learning had been applied to check compatibility of the software modules. In this paper, technique of Boltzmann learning is applied to analyze compatibility of the software modules. The simulation results show that Boltzmann learning performs well in terms of compatibility testing as compared to knowledge learning.

6. ACKNOWLEDGMENTS

This work has been done under the guidance of Assistance Professor Amanpreet Kaur, Chandigarh University. I thanks her for her contribution towards this work.

7. REFERENCES

[1] HERZUM Peter, SIMS Oliver, *Business Component Factory*, Wiley, 1999.

[2] Murat Güneştaşa "Study On Component Based Software Engineering" *A Master's Thesis In Computer Engineering*, Atılım University, JANUARY 2005, pp. 1-97.

[3] Kuljit Kaur Chahal, Harpeep Singh, "A metrics Approach to Evaluate Design of software Components", *IEEE International Conference on Global Software Engineering*, 2008.

[4] Vescan A. "A Metric-based Evolutionary Approach for the Component Selection Problem," *UKSim 2009: 11th International Conference on Computer Modelling and Simulation*, pp. 83-88, 2009.

[5] Arvinder Kaur Kulvinder Singh Mann, "Component Based Software Engineering", *International Journal of Computer Applications*, Volume 2, No.1, pp. 105-108, May 2010.

[6] Vidhu Bhardwaj, "Estimating Reusability of Software Components Using Fuzzy Logic", *A Master's Thesis In Mathematics and Computing*, Thapar University, July 2010, pp. 1-37.

[7] P.G.Chaitanya, Dr. K.V.Ramesh "Feasibility study on Component based software architechure for large scale software system" *International Journal of Computer Science and Information Technologies*, Vol. 2 (3) , 2011, pp. 968-972.

[8] Bhavna Katoch , Rupinder Kaur, "A Review On Maintainability Of Object Oriented Design Metrics", *international journal for advance research in engineering and technology*, Vol. 1, Issue X, pp. 1-4, Nov.2013.

[9] Prasanta Bose ,"Scenario-driven Analysis Of Component-Based Software Architecture Model" *Information and Software Engineering Department George Mason University.*

[10] Gonzalez-Herrera I.; Bourcier J.; Daubert E.; INRIA W. R.; Barais O.; Fouquet F.; & Jézéquel J.M. "Scapegoat: an Adaptive monitoring framework for Component-based systems," *IEEE/IFIP Conference on Software Architecture*, pp.67-76, 2014.