Component Compatibility in Component Based Development

Anitpal Kaur* 
Department of Computer Science and Engineering, 
GNDU, Amritsar, Punjab, India.

Dr. Hardeep Singh 
Department of Computer Science and Engineering, 
GNDU, Amritsar, Punjab, India.

Abstract - This paper presents a review on component compatibility in component based development. Component-based software engineering is a process that emphasizes the design and construction of computer-based systems using reusable software components. Commercial components repositories contain hundred thousand components that make component selection an extremely difficult and time expensive task. Often component selected by functional features are incompatible or the integration effort required is too high. Adding a selection of components based also on compatibility can simplify the integration task. This work focuses on the study of component compatibility using various metrics as parameters.

Keywords- Component Based Software Engineering, Components, Compatibility, Commercial Components, Software Reuse.

I. INTRODUCTION

Primary role of component-based software development is to address the development of systems as assembly components. The developed components are reusable entities, and Component–based development facilitate the maintenance and upgrading of systems by customizing and replacing components. The main advantages of Component based development are reduced development time, cost and efforts along with several others. These advantages are mainly contributed by the reuse of already built-in software components. Generally a component can be defined as an independent and replaceable part of a system that fulfils a clear function. It works in the context of well defined architecture and can communicate with other components through its interfaces. [2]

A. Component

A component is an independent and replaceable part of a system that fulfils a clear function. Components are categories as in component based systems. Conceptual components are the components at the analysis and design phase. Implementation components such as source code files, data files etc. Deployment components involved in an executable system, such as dynamic libraries and executables. It was motivated by the frustration that objects oriented development had not led to extensive reuse as originally suggested. Components are more abstract than object classes and considered to be stand-alone service provider.

B. Component Compatibility

Compatibility of two components is determined by three requirements firstly present the same operational interface to its environment, secondly component should be able to read and write the same data and conform to the semantics in all interactions in which they are engaged. The replacement component’s provided features should obviously be at least the same as those of the current one, otherwise its clients will not be able to successfully communicate with it. Component compatibility defines two levels. Strict compatibility requires that replacement component should be a subtype of the current one. Relaxed level compatibility referred as contextual compatibility because it takes into account two important aspects of the environment in which the current component is deployed. One aspect is which of the current component’s
provided features are actually used by other components in the given application configuration. Secondly requirements of the current component are not considered. [11]

C. Software Component Reuse

Software component reuse can be seen as a process of using available software resources to achieve or update the software system. The available software resources include design, documentation and code and so on. Reused software component is more accurate than new software component because reusable software component already has been tried and tested in working system. There are two approaches for reuse of component, develop the component from scratch or identity and extract the reusable component from already developed component. Reusing software can speed up system production because both development and validation time will be reduced. Software component reuse is a strategy that increases productivity and quality. [1]

II. RELATED WORK

Beibei, Xu. et al. (2012) [1] in paper “Research on software reuse methods based on the object-oriented components” has described component technology as the core technology to support software reuse. Software reuse is considered as an effective method of avoiding duplication of work, improving the efficiency and quality of the software development. The relationship between object-oriented methods and software reuse has been presented. The method based on the component in the object-oriented paradigm is very important to realize software reuse technology. The process of extracting components in order to realize software reuse has been discussed. Software reuse has a great importance in the generality, maintenance and flexibility of development system. Kakarontzas, G. et al. (2012) [6] in paper “Extracting components from open source the component adaptation environment (COPE) approach” has represented open source software an extremely valuable resource that is reused systematically almost in every software project. The reuse of open software components is restricted to ready-made
components and developers who want to reuse code that exists in open source software projects but is not offered as a black-box component often resort to copying existing code and adapting it in their projects. The component adaptation approach for component extraction seems to be effective and easy to use because it is not entirely automated it requires the participation of a knowledge expert.

Cai et al. (2000) [2] in paper “Component based software engineering: technologies, development frameworks and quality assurance schemes” has addressed current component-based software technology, their advantages and disadvantages and the features they inherit. Quality assurance issues for component-based software has been addressed which covers component requirements analysis, component customization and system architecture design, integration, testing and maintenance.

Chengjion, W. et al. (2012)[9] in paper “The reusable software component development based on pattern – oriented” has thought software reuse as a key strategy for reducing development costs and improving quality. Pattern is a way of reuse abstract knowledge about a recurrent problem in a particular context and its solution. The simplicity of a pattern and its small size make it easy to understand, integrate and reuse. It is necessary to link every low level component to the higher level architecture, whereby developers can trace which components satisfy business goal or system requirements because developed software components give arise to the problem that developers do not know why and how they get there. Component model is defined in structured and formal way; the trace is expressed as composition of links. The semantics of links is helpful for developers to reason about the tasks performing with the linkage.

Dusznyski, S. et al. (2011) [4] in paper “Analyzing the source code of multiple software Variants for reuse potential” has presented a scalable reverse engineering technique for assessing the reuse potential among multiple system variants. The technique provides an overview of commonality distribution in the whole analyzed system family and allows for detailed goal-driven refinement of the analysis results. Variant analysis, a scalable reverse engineering technique that aims at delivering exact information. It supports simultaneous analysis of multiple source code variants and enables easy interpretation of the analysis results.

Frankes, W. and Terry, C. (1996) [14] in paper “Software reuse metrics and models” has suggested six types of metrics and models. As organizations implement systematic software reuse programs to improve productivity and quality, they must be able to measure their progress and identify the most effective reuse strategies. This is done with reuse metrics and models. [14]

Mahmood, A. K. et al. (2011) [9] in paper “A mixed method study to identify factors affecting software reusability in reuse intensive development” has considered the benefits of software reuse like reduction in cost, effort and time. Therefore this work focuses on the components compatibility when we reuse entire or some parts of the developed software.

III. FUTURE WORK

In near future this work will be extended to purpose a new component based system method using some decision tree algorithm to have the decision making criteria.

IV. CONCLUSION

This research work has focused on the components compatibility when we reuse entire or some parts of the developed software. Software reuse is an important technology which can avoid repeated labor, improve quality and productivity. As component base development reuse components from different software’s at a time but each software engineer has used its own naming conventions and methods so it becomes difficult to reuse the software directly. So it is required to convert the components into reusable form. Thus component compatibility becomes important part in component based development. Therefore this work focuses on the components compatibility which occurs during the software reusing time. The overall goal of this work is to select only those components which come up with optimal values when they are going to reuse.

REFERENCES


