ISSN: 2319-7080 International Journal of Computer Science and Communication Engineering
Volume 2 Issue 1 (February 2013 Issue)

Analysis of Component Composition Approaches

Pooja Soni[1], Nisha Ratti[2]
[1]M-Tech Scholar, [2]Assistant Professor
[1, 2] Department of Computer Science, Rayat Institute of Engineering & Information Technology, Railmajra (Punjab), India
[1]poojasoni.14388@gmail.com

***Abstract*-Component-based software engineering is increasingly being adopted for software development. This approach, which uses reusable components as building blocks for constructing software, can facilitate fast-paced delivery of scalable, evolvable software systems. A component-based software system often consists of a set of self-contained and loosely coupled components allowing plug-and-play. The process of assembling components to create a system is called Component Composition. Composition is used to build complex components from simpler ones. This paper illustrate the different approaches of component composition and differentiate them on the basis of different parameters so that it would become easier to select appropriate approach for component composition in particular software system.**

***Keywords*-Component, Component-based software engineering, Composition**

1. Introduction

The increasing importance of introducing component-based software engineering principles early in introductory undergraduate computer science education is widely recognized[12]. While there is still considerable debate among educators on the principles to be taught, there is reasonable consensus that students should be educated (at least) on the following fundamental software development principles:

• Independent software development: Large software systems are necessarily assembled from components developed by different people. To facilitate independent development, it is essential to decouple developers and users of components through abstract and implementation-neutral interface specifications of behaviour for components.

• Reusability: While some parts of a large system will necessarily be special-purpose software, it is essential to design and assemble pre-existing components (within or across domains) in developing new components.

• Software quality: A component or system needs to be shown to have desired behaviour, either through logical reasoning, tracing, and/or testing. The quality assurance approach must be modular to be scalable

.• Maintainability: A software system should be understandable, and easy to evolve.

Composition of components is the process which follows all the fundamental principles of component based software development. Composition systems are those systems that concentrate on composition of components. They generalise many approaches to component based engineering approaches. Component composition approaches are defined through a component model, component technique and composition language.

In this paper there is analysis of component composition approaches. A large number of components need to be assembled together for a new system and interaction among these components is quite complicated and sophisticated, so corresponding measures have to be taken during composition process. There are various approaches available for composition of software components. The nature of components is not always the same. Components appear on different granularity levels, deals with different stakeholder requirements, or are simple design concepts. There are various types of components.

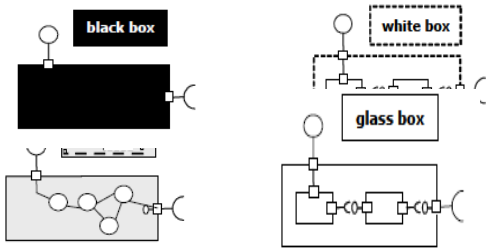


Figure1 Four types of components

Black Box Components: Black box components are typically in compiled or binary form. They are discrete components that cannot directly be changed. All the programmer knows about them is the documentation that describes their functionality, and their published "publicly known" interfaces[2].These interfaces may include properties (or attributes) that can be viewed, or new values placed in them. Black box components cannot directly be modified by a programmer.

White Box Components: White box components are components that are source code. They are readable, and directly changeable by the programmers that use them.White-box description defines exactly what the system does, it is over-specified. An ideal description should tell us only as much as is required to understand the system, but no more than is necessary; Abstraction is the key.

Grey Box Components: The Grey-box components stands in the middle ground between black- and white-box components. A grey box reveals parts of its internal workings, not just relations between input and output. The information can become as detailed as necessary where needed, for instance, to state under what conditions external components are called [9]. In other places it may remain very abstract and simply state a condition that is established.

Glass Box Components: The glass-box view attempts to retain the advantage of the purely declarative representation (expressed in terms of truth conditions) found in the blackbox view. The aim is to provide support for meta level reasoning about the process (i.e rather than just executing it) by exposing the rules that govern it. Put another way ,a glass-box view exposes what the process does, without necessarily giving away it how it does it.

II. Composition Approaches

Composition approach describes how various types of components be joined together to work efficiently. As there are different types of components the composition approaches are also different. For building a particular composition system a composition approach is required whose selection depends upon the type of components are going to be used in component system. There are basically two types of component composition approaches i.e black box composition approach and grey box composition approach. Black box components typically describe what can be done,rather than how it can be done. They represent the pieces of functionality that can be combined. Black box components can not be directly modified by a programmer. Traditional composition approaches are characterised by applying black box components. As compared to black-box components, white box components allow the direct access to the internal details and the programmer can modify the component as well [12]. So there are no approaches for the composition of components. Compared to black box components, Grey-box components are opened for structural changes. Grey-box components introduce a higher level of parameterisation and therefore wider adaptability and reuse.

According to ABmann, all software composition systems can be compared in terms of three major aspects which form a composition system of a:

1. Component model that answers the question," How do components appear and when can they be exchanged?"

2. Composition technique that answers the question, "How components composed?"

3. Composition language that is needed to write composition specifications. These specifications describe how systems should be built from components and contain information about their architecture.

1. Black box composition approaches

Black box approaches have already been successfully applied in industry for many years. There are approaches for black box composition and these approaches are: modular composition, object oriented composition, component-based composition and architecture composition. Each approach introduced new concepts that improved the quality of the process of composition and of the resulting products.

1) Modular Composition

Parnas and Dennis were one of the initial contributors to the topic. Parnas et al. Discussed the modularisation as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. A module is a work assignment for a programmer or programmer team. Each module consists of a group of closely related programs. Modules introduced encapsulation – combining data and behaviour in one package and hiding the implementation of the data from the user of the object. Coding techniques in the area of modular programming results in such major advancements as(1) low coupling and (2) replacement. Low coupling is expressed by the fact that a module can be written with little knowledge of the code in another module. Modularisation allowed for more flexible replacement and reassembling of modules without the reassembling the whole system. Technically, a module is a unit of source code that can define constants, data types, variables, functions and procedures. A module can be compiled independentaly of other code. With an export declaration, a module can allow access to its internal entities by gaining access to another module. Modular systems can be flexible recombined during design time. However, during run time, modular system remains static, as it is not possible to create and link new modules dynamically.

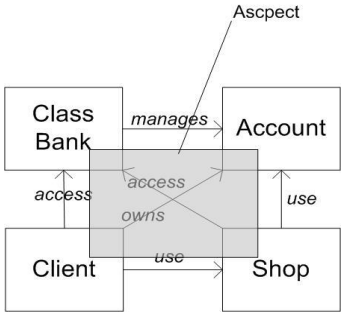
2) Object-Oriented Composition

Object-oriented software composition was introduced in order to simplify the development of software systems and to make it possible to change the system dynamically. Mainly, objects were introduced as elements of software composition during the development of Stimula 67 programming language, which was influential in the development of later object-oriented software composition models. The modular concept was enhanced and new elements of composition, so called objects, were defined. In the object-oriented composition, important new terms were defined. These are class, object, polymorphism and inheritance. In contrast to modules, it is easier to describe real world problems with objects. Classes are specification of a set of objects with similar behaviour. Objects are created according to the class specification and interact with some other objects that constitute a software systems. Specifications encapsulated in classes can be further extended with the help of inheritance.

An object-oriented composition model provides major advantages over modular composition, such as dynamic extension of the system and a more flexible and comprehensive way of describing a target system. However, composition with an object is an identity-based approach, as the identity of an object is a criteria for required behaviour, a further step over the object-oriented approach was the composition of software systems with components.

3) Component-Based Composition

Component based composition stands for designing software systems from application elements that were constructed independently by different developers using different languages, tools and computing platforms. In comparison to objects, components alone are represent stand-service providers.There different definitions:



Szyperski "*A software component is a unit of composition with contractually-specified interfaces and context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*"

Somerville[8] *" Components provide a service without regard to where the component is executing or its programming language:*

A component is an independent executable entity that can be made up of one or more executable objects;

The component interface is published and all interactions are through the published interface."

There are many different standards defined for components. A component model is a definition of standards for component implementation, documentation and deployment. The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

4) Architecture-Based Composition

Architecture-based composition introduces a novel composition model with connectors and ports, abstracting communication routines from the component implementations. Connectors are modules for communication. Ports are connection points indicating when data flows in and out of a component. In order to help the architectural designers with their tasks, the formal notations for representing and analyzing architectural design were proposed. These notations are referred to as "Architecture Description Languages".

Clements specified ADls as forml languages that can be used to represent the architecture of a software-intentive system.

The communication mechanism in the architecture-based composition is encapsulated and represents a parameter through which a system can be adapted to a range of newly defined requirements. This results in a wider applicability of components. Taking this into consideration, it is possible to say that the architecture-based composition is a step in the direction in the direction of using grey-box elements and the aspect-oriented composition models.

2. Grey-box composition approaches

Compared to black-box approaches, grey-box approaches apply grey-box components as units of composition. The typical example of a grey-box composition approach is the AOP.In this approach, a grey-box component is an aspect. Aspects maybe weaved by demand into other components[9],[12].The grey-box composition approaches going to be reviewed are aspect oriented programming, composition approaches based on multi-dimensional separation of concerns, feature-oriented composition and invasive software composition.

1) Aspect-Oriented Programming

Aspect-Oriented Programming is a method developed at the Xerox Palo Alto Reasearch Center by Gregor Kiczales.AOP allows programmers to first express each of a system's aspects of concern in a separate and natural form, and then automatically combine those separate descriptions into a final executable form using a tool called an Aspect Weaver. AOP addresses the issues related to the decomposition process. Many a times, during the process of decomposition, some non-functional concerns cannot be taken care of. The code of these cross-cutting concerns, span across many modules which in turn have adverse effect on the quality of the software

Figure2 Modules and Aspects

The above figure shows a concern which cuts across target modular decomposition. In AOP, aspectual decomposition techniques are used rather than functional decomposition . An important difference between aspects and functional units is that aspects fundamentally cross-cut both each other and the resulting executable code. As such a cohesive aspect of

concern in the source program ends up being spread about and mixed in with other aspects in the output of the weaver.

Some of the useful component aspects are user interface, collaboration, persistency, distribution, and configuration. Unlike traditional object-oriented analysis object services, aspects may share component services, required aspects are as important to characterize as provided aspects, and often more than one other component may provide or require a component's aspects.

2) Composition Based on N-Dimensional Separation of Concerns:

This approach is based on N-dimensional separation of concerns. Unlike black-box approaches, which have only one kind of concern, all the other grey box approaches have a number of concerns to be taken care of.
Tarr et al. specified a related term,"tyranny of the dominant decomposition"

Existing formalisms at all lifecycle phases provide only a small, restricted set of decomposition and composition mechanisms, and these typically support anly a single, "dominant" dimension of separation at a time.

In order to solve the problem of the Tranny of the dominant decomposition, a strategy called Multi-dimensional separation of concerns (MDSoC), is introduced. It is based on the support for simultaneous separation of overlapping concerns in multiple dimensions.

The term multi-dimensional separation of concerns to denote separation of concerns involving [6]:

- Multiple, arbitrary dimensions of concern.
- Separation along these dimensions simultaneously.
- The ability to handle new concerns, and new dimensions of concern, dynamically, as they arise throughout the software lifecycle.
- Overlapping and interacting concerns; it is appealing to think of many concerns as independent or "orthogonal," but they rarely are in practice.

The MDSoc is a new, fresh strategy and it opens the door for further research questions to be investigated. Basically, it is not yet integrated into the real working industry solutions[17].Next, we will shortly review subject-oriented programming as an initial step in the direction of MDSoC, followed by hyperspaces, as one of the currently proposed approaches for MDSoC.

The SOP was introduced by **Ossher et al.**[18].The SOP supports the packaging of object-oriented systems into subjectsThe SOP provides a means for composition designers to write composition expressions.A subject is defined as a mean a collection of state and behavior specifications reflecting a particular gestalt, a perception of the world at large, such as is seen by a particular application or tool. Hyperspaces permit the explicit identification of any concerns of importance, encapsulation of those concerns, identification and management of relationships among those concerns, and integration of concerns . The main concepts of hyperspace programming are units, hyperspaces, hyperslices, and hypermodules. A unit is a code fragment of the underlying language, e.g. a class or a member definition. Hyperslices are a set of units that are declaratively complete in order to eliminate coupling between hyperslices. A compound set of hyperslices is called a hypermodule. They can be composed to larger hypermodules until the final system results. The major advance over SOP and AOP is the ability to simultaneously support the clean separation of multiple different kinds of potentially overlapping concerns, with on-demand re modularization [8].

3) Feature Oriented Composition

A feature in software composition represents a certain piece of functionality.**Batory et al**.[20] defines a feature as a characteristic that programs of a product-lone can share; distinct programs in a product-line are described by distinct combinations of features. Prehofer [9] proposed a new model for the flexible composition of objects from a set of features. This model was called Feature-Oriented Programming (FOP). Unlike object- based composition, the feature-based composition made it possible to create objects with individual services by selecting the desired features. Like the Aspect-Oriented composition, the FOP was developed in order to overcome problems that appeared when using classical decomposition techniques like OOP. These problems include the presence of crosscutting concerns. The FOP modularizes crosscutting concerns and is based on collaboration design and refinements and can be classified as a part of the MDSoC concept [8].

4)Invasive Software composition

In this approach, software components are composed by program transformation. In Invasive Software Composition (ISC), a component is a program fragments or elements. A fragment box is a set of program elements. A fragment box has a composition interface that consists of a set of hooks.[19].A hook is a point of variability of a fragment box, a set of fragments, or positions that are subject to change. A composer is a program transformer that transforms one or more hooks for a reuse context. Composers instantiate, adapt, extend, and connect fragment boxes by transforming their hooks. Fragment boxes are a collection of code templates. Fragmented boxes are parameterised with hooks. There can be of two kinds of hooks- a code hook and a position hook. Hooks containing only program elements are called code hooks. Position hooks are those which consist of positions in the component's code.

Invasive software composition not only provides a full-fledged composition language, but also offers a grey-box component model for the construction of tightly connected systems. It enables generic programming, connectors in a standard language, inheritance calculi, view-based programming, and aspect-based development. Invasive

software composition also has some restrictions. It is a code-based component technology, i.e. it works at compile-time and require classes.

III. Comparative Analysis of Black-Box & Grey Box Composition Approaches

Following parameters are taken as the basis of comparison between the above discussed two approaches.

Abstraction: In case of Black-box composition, level of abstraction is higher as compared to grey box composition. All the internal details are not revealed to the end user as in case of Grey-box compositions.

Structural Transformations: Grey-box components can be easily changed in terms of structure and behaviour. But in case of black-box components, no changes can be done by the user.

Extensibility: Black-box components are easily extensible in nature while grey-box components allow the extensibility up to some limit only.

Customizability: As black-box components do not reveal the internal structure as they cannot be customized. But Grey-box components allow the customization of the components.

User-Interaction: User cannot interact with the internal details of the black-box components. Only input can be submitted to the black-box components and output can be seen as a result of the functioning of the components. Unlike black box components, grey-box components allow better user interaction

Table1. Comparative analysis of component composition approaches

Parameters\ Approaches	Black-Box Composition	Grey-Box Composition
Level of Abstraction	Higher	Lower
Structural Transformation	Not Done	Easily Done
Extensibility	Easy	Limited
Customizability	Not Allowed	Allowed
User-Interaction	Not allowed	Allowed

IV. Conclusions

In this paper analysis of component composition approaches has been done. Different composition approaches such as modular composition, object-oriented composition, component-based software composition, architecture-based composition are studied with respect to black-box composition approach.With respect to Grey-box composition different approaches are aspect oriented programming, composition approaches based on multi-dimensional separation of concerns, feature-oriented composition and invasive software composition. A comparative analysis on the basis of these two approaches is also performed. Various parameters like level of abstraction, structural transformations, extensibility, customizability and user-interaction are taken for their study. It is found that black-box components do not allow structural transformation, customizability and user-interaction as compared to grey-box components, whereas they allow easy extensibility because they support higher level of abstraction. On the other hand, Grey-box components support limited extensibility along with lower-level of abstraction. Structural transformations, customizability and user-interactions can be easily done with respect to Grey-box components.

References

[1] Hans de Bruin, A Grey-Box Approach to Component Composition, Vrije Universiteit, Amsterdam, The Netherlands,1999

[2] Steve Battle, Boxes: Black, White, Grey and Glass box views of web-services, Digital Media Systems Laboratory HP Laboratories Bristol HPL-2003-30February 24th , 2003

[3] I. Crnkovic and M. Larsson (Eds.). Building Reliable Component-Based Software Systems. Artech House Publishers, 2002. ISBN: 1580533272

[4] Murali Sitaraman. Timothy J. Long. E. James Harner, A Formal Approach to Component-Based Software Engineering:Evaluation and Education,2001

[5] Manju Kaushki and M. S. Dulawat, A Comparision Between Traditional and Component Based Software Development Process Models,Research Scholars, Mohanlal Sukhadia University, Udaipur, INDIA. Department of Mathematics and Statistics, Mohanlal Sukhadia University, Udaipur, INDIA,May 2012.

[6] Clemens Szyperski,Domnik Gruntz,Srephen Murer Component Software Beyond Object-Oriented Programming ,Second Edition, ISBN 0-201-74572-0 2nd edition.

[7] OMG, CORBA, http://www.omg.org/corba

[8] I. Sommerville. Software Engineering (6th Edition). Pearson Studium, Germany,2001. ISBN: 3827370019

[9] Buchi M. and Weck W., A plea for grey-box components, Turku Centre for Computer Science, TUCS Technical Report No 122, August 1997, ISBN 952-12-0047-2, ISSN 1239-1891

[10] Software composition, http:// www.win.tue.nl/~mchaudro

[11] Component Models, Component-Based Software Engineering, www.win.tue.nl/~mchaudro/cbse2005

[12] Yermashov K., Software Composition with Templates, DeMontfort University, Ph.D.Thesis, June 2008

[13] OMG, UML, http://www.uml.org/

[14] Casandra Holotescu, Black-Box Composition: a Dynamic Approach,Department of Computer and Software Engineering Politehnica University of Timi ,soara, Romania,2010

[15] K. Sergei and Lorenz D.H., Comparing White-Box, Black-Box, and Glass-Box Composition of Aspect Mechanisms,M. Morisio (Ed.): ICSR 2006, LNCS 4039, pp. 246–259, 2006

[16] A. Cechich, M. Piattini, A. Vallecillo,"Assessing Component-Based Systems", Component Based Software Quality, LNCS 2693, pp. 1-20, 2003

[17] Microsoft, "The Component Object Model: A Technical Overview (MSDN)", http://msdn.microsoft.com/en-us/library/ms809980.aspx

[18] William Harrison and Harold Ossher. Subject-Oriented programming: a critique of pure objects. In OOPSLA'93:Proceedings of the eighth annual conference on Object-oriented programming systems,languages, and applications,New York,NY,USa,1993.ACM Press

[19]U.Abmann.Invasive Software Composition. 540443851

[20] Microsoft, ".NET Development (MSDN)", http://msdn.microsoft.com/enus/library/aa139615.aspx