Volume 4, Issue 9, September 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Component Based Software Engineering – At a Glance

[1]Maushumi Lahon, [2]Uzzal Sharma

[1] Department of CE and Application, Assam Engineering Institute, Guwahati, India

[2] Department of CS and IT, Don Bosco College of Engineering and Technology, Assam Don Bosco University, Guwahati, India

Abstract : Methods for building software has come a long way from traditional to structured then to object oriented concept and now component based software engineering (CBSE). But when we talk in terms of building the same, the methodology was similar i.e. to build from the scratch until the concept of components evolved and development of software was thought of in terms of assembling the components. There are a lot of literatures regarding the component based software development which covers many aspects like components, interfaces, developing environment, component models, their construction, issues in CBSE, challenges in CBSE, design issues in components and many approaches and models to bridge the gap between the existing CBSE and the ideal CBSE. This paper intends to cover certain aspects in the field of CBSE and present the reviewed material in a systematic manner to provide an overall view about the various aspects of CBSE. The aim is to some extent summarise the knowledge of the wide range of areas to be addressed in this domain of software engineering.

Keywords: CBSE, Components, Component Model, Component Framework, Object Oriented Paradigm.

I. INTRODUCTION

The paradigm shift from building software from scratch to the engineering of assembling components to build software involves many issues and it demands more and more precision in every aspect to make this new paradigm as the most convenient way to resolve the software crisis. CBSE attempts to answer the question “Can a software be built by assembling components like assembling a car ? “ Following the success of the structured design and OO paradigms, Component-Based Software Engineering (CBSE) has emerged as the next revolution in software development. CBSE evolved as a popular methodology in the early 90s and was expected to produce high quality software at reduced cost and lesser time as development of software need not be done from scratch. The concept of reusability was considered as the backbone of CBSE. This methodology started becoming popular after the introduction of middleware technology support like Microsoft’s Component Object Model (COM), CORBA, and JavaBeans etc.

II. A WORD ABOUT COMPONENT BASED SOFTWARE ENGINEERING (CBSE)

The field of Software Engineering has come a long way from traditional development where the design was dependent on the knowledge; experience and understanding of the programmer (developer) to the present object oriented development methodology which in turn forms the basis of the concept of reusability and CBSE. The concept of structured programming provided the actual shift from traditional development to a systematic approach to software development which provided specific areas for research and bring Software Engineering to this present day state. From the first NATO software engineering conference held at Garmisch, Germany, on October 7-11, 1968 this is the 46th year of Software Engineering. “According to ANSI/IEEE standard 610.12-1990, software engineering was defined as the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software ” [1] .

CBSE emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific whereas components are more abstract than object classes and can be considered to be standalone service providers. In 1990’s the development of CBSE was also around the concept of other engineering disciplines trying to find out the systematic approach to build software by assembling the various components. The major goals of CBSE are the provision of support for the development of systems as assemblies of components, the development of components as reusable entities, and the maintenance and upgrading of systems by customising and replacing their components [2]. The concept of creating software by picking up available components which could fulfil the user requirements gave rise to the field of Component Based Software Engineering. According to a report “The Component-Based Software Engineering (CBSE) Market Assessment,” this field was expected to grow in the next 5 – 10 years from the publication of the report (companion report published along with the report in reference [3] but due to lack of agreement among analysts, researchers, technology producers, and consumers about what software components are, and how they are used to design, develop and field new systems [3] the predictions were tainted. Though the expected growth could not be achieved at the predicted rate but there is a growth at slower pace and is still in the growth phase with researchers, technology producers, analysts and consumers working in the field of designing components, developing specifications, component composition, developing interfaces, extracting reusable components and other related areas of CBSE. Scope exists in this area for further work to make the field attain maturity as in case of other component oriented approach applied to other fields of engineering. The CBSE process can be represented as a flowchart given in the Figure 1.

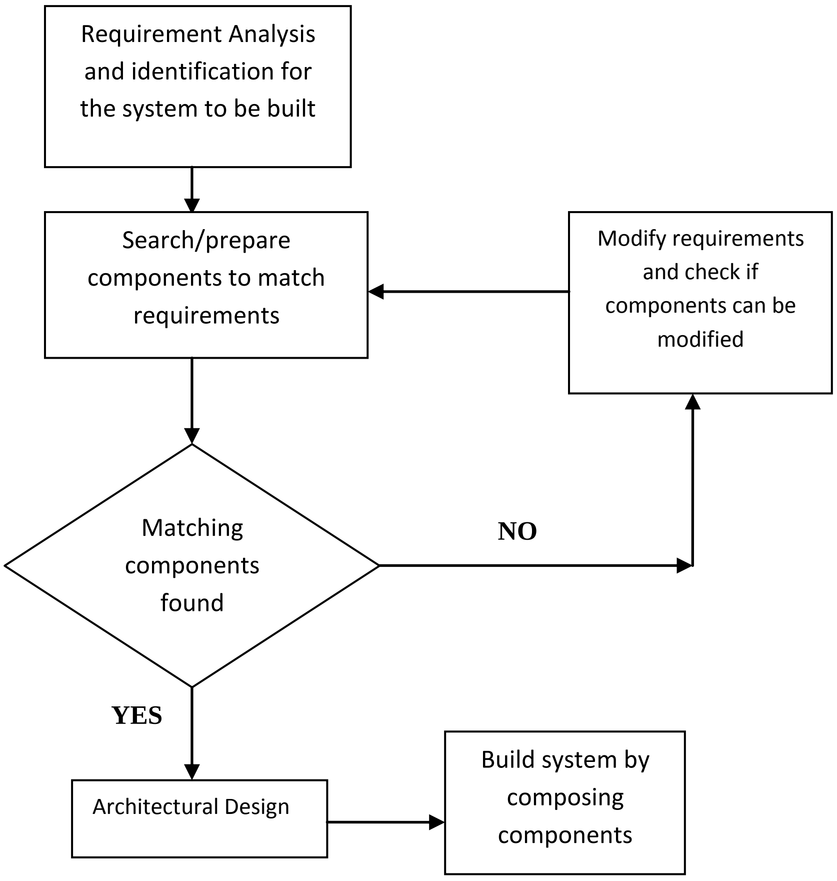
Requirement Analysis and identification for the system to be built
Search/prepare components to match requirements
Modify requirements and check if components can be modified
Matching components found
NO
YES
Architectural Design
Build system by composing components

Fig 1: CBSE process

III. FUNDAMENTALS OF A COMPONENT

There are many ways a component is defined and each definition covers a different aspect in defining the component. Components are well established in all other disciplines but until 1990's they were unsuccessful in the field of software engineering. Software components are executable units of independent production, acquisition and deployment that can be composed into a functioning system. Components are independent and do not interfere with each other and their implementations are hidden. Communication in components are through well-defined interfaces and component platforms are shared which reduces development costs. According to Szyperski [4], "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties". Another definition was given by Meyer [5] as : "A component is a software element (modular unit) satisfying the following conditions:

1. It can be used by other software elements, its 'clients'.
2. It possesses an official usage description, which is sufficient for a client author to use it.
3. It is not tied to any fixed set of clients."

In his book [4], Szyperski says that components are for composition and to enable composition software components adheres to a particular component model and targets a particular component platform. This concept of component model was also found in the definition of component given by Heineman and Councill [2] as: "A component is a software element that conforms to a component model and can be independently deployed and composed without any modification according to a composition standard."

The different definitions given by different authors thereby define a component from different perspectives but the following points are distinct from the various definitions:

1: It is an independent entity;
2: It is a reusable entity;
3: It should conform to some component model ;

According to [3] the characteristic properties of a component are :

- It is a unit of independent deployment;
- It is a unit of third party composition;
- It has no (externally) observable state;

In the survey conducted by the Manchester University [8], a component is considered to be a software unit consisting of a name, an interface and code. Interface forms the only way of communicating with a component. Every component must have interfaces which have a standard that declares what an interface should comprise of.

From the literature found in [2], [3], [4], and [6] it is commonly accepted that a component has a life cycle which consists of three phases – design, deployment and runtime. In the design phase the components are constructed in their source code and kept in the repository. Components in this phase cannot be executed. In the deployment phase binaries of the component are created and deployed into the target system and in the runtime phase the binaries are instantiated with initial data.

IV. THE PURPOSE AND DEVELOPMENT OF COMPONENT MODEL

A component model specifies the standards and conventions imposed on developers of components. Though there is no universally accepted agreement as to what is to be included in the model, the purpose of a component model or the expectation for introducing the conventions imposed on developers can be identified. According to the SEI technical report [3] the aim is to attain -

- Uniform composition i.e. two components can be combined to communicate when both share the same assumptions,
- Ensure quality attributes i.e. standardising components used and their pattern of interfaces will ensure quality in case of system composition from third party components
- Deployment of component and composition i.e. components must be able to be deployed from developers to composition environment and applications from composition to customer environment.

A component model is considered to define specific interaction and composition standards. It defines how to construct an individual component as well as set behaviour as to how components or set of components will interact with each other. A component model implementation on the other hand is the set of executable software elements required to support the execution of components that conform to the model [3]. A component model provides two types of services [7]:

- Horizontal services which includes component management, transaction management, resource management, concurrency, persistence and security.
- Platform services which includes addressing, interface definition, exception management and component composition.

To use services provided by a model, components are deployed in a container which is a set of interfaces used to access the service implementations. In a survey on components conducted by University of Manchester [8] the components were surveyed on the basis of an abstract framework in which the three aspects syntax, semantics and composition of the components were considered.

I. Syntax referred to the language in which the component is constructed,
II. semantics specified what the component is meant to be in terms of its required and provided services in the form of interfaces and
III. composition which is vital issue as components are designed for composition are expressed in terms of a composition language.

The survey concludes that the current models uses connectors or glue code, UML notation, , (ADL) [9] and ADL like languages CoCo and CCL [10] for composition.

Current component models can largely be divided into two categories [8], [11]:

1) Models where components are objects, as in object-oriented programming,
2) Models where components are architectural units, as in software architectures [12], [13]

According to the survey conducted by Lau and Wang [14] on thirteen component models presently available, it was found that those models where components are classes, the definition language for components are same as their implementation language, but in cases where components are architectural units, the definition language is an ADL or an ADL-like language and their implementation language either do not exist like in Acme [15] or have a different implementation language like in Koala. This study also reveals that there is no specific composition language for all models, some like EJB [16], [17] do not have any composition language whereas Koala [18], [19] uses connectors. Hence to reason out composition i.e. what will be the output of a composition a composition theory is necessary. Current component models lack in this area. The study also categorises the current component models on the basis of syntax, semantics and composition.

Expanding the horizon of component models Lau and Taweel [20] proposed that domain specific component models could give more productivity and also argued that the present component models though intended to be domain specific are not actually domain specific. The paper presents an approach to derive a domain specific component model from the domain model of the domain and also establishes that domain specific components are better. As domain modelling has a more or less standard terminology the author has used the same in the paper. The work focuses on the functional and feature sub – models of domain modelling. Lau et.al in 2011 [21] also proposed a model beneficial for industries which is both control driven and data driven and can be considered as a component based counterpart of the existing tools and languages existing in the industries. The proposed model allows for separation between control flow and data flow which leads increased reuse, modularity and more structured architecture.

Component models are executed in a particular environment which supports the components that follow that component model. This can be termed as the Component framework. As processes are run in the Operating System (OS), components communicate through the component framework. It can be considered as a layer above the OS for enabling interaction between components. The Enterprise Java Beans(EJB) defines a framework to support EJB component model, WaterBeans framework supports visual composition of components and Microsoft's VisualBasic framework called VBXs is one of the most successful commercial framework.

V. CHALLENGES IN CBSE

CBSE presents a promising paradigm in software development but it is yet to attain maturity in terms of product life cycle terminology which consists of initiation, growth, maturity and decline. As CBSE is in the growth phase there are many challenges which need to be addressed. The analyses conducted during the Component Based Software Engineering network (CBSEnet) work by a panel of several experts identified the challenges (using CBSE Classification Model) and the gap between the ideal situation and the actual practice were listed [22]. The gap was identified in terms of research, technology and standard and the CBSE aspects were identified as concepts, process, business, product, technology, off the shelf components and related paradigm. The main challenge in this domain is achieving a best conversion procedure from requirements to components and then from components to system. As requirements vary from one client to another, it becomes difficult to understand and build the component exactly to fulfil the requirement and so reusing a component always require some amount of modification to exactly fulfil the requirements. Other challenges in this area includes the issue of reliability in terms of deliverables of the component, time and effort in developing components, unclear and ambiguous requirements, component maintenance cost, conflict between usability and reusability etc.

VI. CONCLUSION

CBSE is projected as the future of software engineering and so more and more areas are addressed in this field. This field has more challenges because of lack of standardisation. As there are many platforms and requirements of one do not exactly match with another, components designed cannot be reused exactly the way it is made and hence needs modifications. Moreover, areas related to component composition also require more of standardised composition theory which can be implemented both in object oriented components and components based on architectural units. Services provided and services required by components need to be expressed in standard form so as to enable maximum reuse which will also improve trustworthiness of components. This will provide more reliable components and there will be more and more acceptance of this domain in building software. This in turn may lead to lesser development time and cost.

REFERENCES

[1] Wang J A, "Towards Component-Based Software Engineering", Rocky Mountain Conference, CCSC, 2000

[2] Heineman G T , Councill W T, editors "Component Based Software Engineering : Putting the pieces together ", Addison Wesley, 2001

[3] "Technical Concepts of Component-Based Software Engineering",Vol-II, CMU/SEI-2000-TR-008

[4] Szyperski C, Gruntz D, and Murer S, " Component Software: Beyond Object-Oriented Programming", Addison Wesley, second edition, 2002.

[5] Meyer B, " The grand challenge of trusted components", InProceedings ICSE 2003, pages 660–667, IEEE,2003.

[6] Broy M, Deimel A, Henn J, Koskimies K, Plasil F, Pomberger G, Pree W, Stal M, and Szyperski C, "What Characterizes a Software Component?" Software—Concepts and Tools, vol. 19, no. 1, pp. 49-56, 1998.

[7] Sommerville I, "Software Engineering", Chapter 19, 7th edition, 2004

[8] Lau K K and Wang Z, A Survey of Software Component Models,second ed., School of Computer Science, Univ. of Manchester, http://www.cs.man.ac.uk/cspreprints/PrePrints/cspp38.pdf, May 2006.

[9] Allan R , Garlen D, 'A formal basis of architectural connection, ACM transactions on Software Engineering and Methodology, 6(3) 213-249, 1997

[10] Wallnau K C and Ivers J, 'Snapshot of CCL: A Language for predictable Assembly,Technical Report DRAFT/CMU-SEI-2003-TR-009.

[11] Lau K K, "Software Component Models," Proc. 28th Int'l Conf. Software Eng. (ICSE '06), pp. 1081-1082, 2006.

[12] Shaw M and Garlan D, Software Architecture: Perspectives on an Emerging Discipline.Prentice Hall, 1996.

[13] Bass L, Clements P, and Kazman R, Software Architecture in Practice, second ed. Addison-Wesley, 2003.

[14] Lau and Wang, Software Component Models, IEEE, October 2007

[15] Garlan D, Monroe R, and Wile D, "Acme: Architectural Description of Component-Based Systems," Foundations of Component-Based Systems, G. Leavens and M. Sitaraman, eds., pp. 47-68,Cambridge Univ. Press, 2000.

[16] DeMichiel L, Yalc¸inalp L, and Krishnan S, Enterprise JavaBeans Specification Version 2.0, 2001.

[17] Monson-Haefel S, Enterprise JavaBeans, fourth ed. O'Reilly & Assoc., 2004.

[18] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala Component Model for Consumer Electronics Software," Computer, vol. 33, no. 3, pp. 78-85, Mar. 2000.

[19] R. van Ommering, “The Koala Component Model,” Building Reliable Component-Based Software Systems, I. Crnkovic and M. Larsson, eds., pp. 223-236, Artech House, 2002.

[20] Kung-Kiu Lau and Faris M.Taweel, “ Domain-Specific Software Component Models”, G.A. Lewis, I. Poernomo, and C. Hofmeister (Eds.): CBSE 2009, LNCS5582, pp. 19–35, 2009.

[21] Kung-Kiu Lau, Lily Safie, Petr Štˇepán and Cuong Tran, “ A Component Model that is both Control-driven and Data-driven” , CBSE’11, June20–24, 2011, Boulder,Colorado, USA.

[22] Stefano De Panfilis and Arne J. Berre, ‘Open issues and concerns on Component Based Software Engineering’,