ISSN: 0975–833X

Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 3, Issue, 5, pp.075-077, May, 2011

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

REVIEW ARTICLE

PRAGMATIC SURVEY AND COMPARATIVE ANALYSIS OF SOFTWARE ARCHITECTURE UNIFIED MODELING LANGUAGE AND COMPONENT-BASED SYSTEM DEVELOPMENT LIFE CYCLE MODELS

*** Rev. Engr. Dr. Okafor, Friday Onyema**

Department of Mathematics/Statistics/Computer Science, Michael Okpara University of Agriculture, Umudike

ARTICLE INFO

Article History:

Received 11th January, 2011
Received in revised form
18th February, 2011
Accepted 15th March, 2011
Published online 14th May 2011

Key Words:

UML,
COTS,
Component-Based Development,
Reusability
Cyclic life cycle model.

ABSTRACT

Software provides a means for productive activities of man and industries. Indeed the strength, intelligence and vastness of the use of computer lie on the software. Sequel to the above, this paper discusses the component- based system development life cycle, reusability and identifies the acquisition and elicitation of requirements in combination with COTS selection as a multi- criteria decisions process which possess great problem**.** It suggests the use of cyclic life cycle paradigm as a solution to this great problem. The paper compares Component- Based Development cycle with the Cyclic Life Cycle model, and software architecture with component- Based Development and concludes that the component- Based approach and architectural design play vital role in product configuration management. Finally, it compares Unified Modeling Language (UML) and Component- Based systems modeling. The UML can be used for both component and system modeling while the Component- Based design should best concentrate on interface definitions and collaboration between the components through the interfaces. Finally using an appropriate diagram, the paper presents the various aspects of Component- Based Architecture which include conceptual architecture, implementation architecture and deployment architecture.

INTRODUCTION

The erratic and ever- growing requirements of users for advanced functionalities as well as the proliferation of software application in all fields of life, have created high demand for software and have made software development process more complex. Today, software is used in games, fund transfer, weather prediction, space craft guide, data mining, internet operations, vast information storage, music, industries, military, homes, just to mention but a few. The intelligence and vastness of the computer application is dependent on the ability of the computer software to interact with the computer hardware. Thus, the need for critical survey, comparative analysis of the software architecture, Unified Modeling Language and Component Based Development models becomes inevitable. Component based software development (CBD) is being described as the conceptualization of software systems as a composition of individual reusable components that come together to achieve the objective of a system. According to Crnkovic (2001:127-133), component development involves mainly reusability and can be deployed independently and may include a third party software such as commercial off the shelf (COTS) tools. CBD is a high level abstraction of software system development that tolls an approach that is very similar to industrial component-based assembly line procedure.

Corresponding author: revmachi_4@yahoo.co.uk

This approach makes CBD very attractive and helps to reduce software development time and cost drastically. This paper compares CBD with the cyclic life cycle (CLC) paradigm and observes that finding and selecting of components in CLC is replaced with requirements identification and design in CBD, hence component based development requires a detailed analysis and thus, meeting up the requirements becomes very difficult and impossible. This is one of the major challenges that face CBD. It is also important to observe and state that software architecture and components are closely related to each other. This relationship could be seen in the fact that functionality-based architecture design is the first phase of the structuring and then software architecture assessment Osuagwu (2005). More so, once the software architecture is defined, components selection/development will result. UML which is a graphical language for specifying a process is a standard way for writing system blue prints. It is used for both component and system modeling

Component-based system development life cycle

The main emphasis when developing components is reusability, though other components could be incorporated. A component must be well specified, easy to understand, sufficiently general, easy to adapt, deliver, deploy and replace. The component interface must be simple and strictly separated (both physically and logically) from its implementation. In component development, acquisition and elicitation of

requirements in combination with COTS selection is a multi-criteria decisions process and posses great problem. It is indeed highly probable that COTS, meeting all the requirements will not be found. Early selection of components in the development process may not even meet all the requirements. In practice, development with components is focused on reusable entities and relations between them, starting from the system requirements. There are two essential steps involved in the early design process:

1. Specification of a system architecture in terms of functional components and their interaction-this gives the logical view of the systems and
2. Specification of a system architecture consisting of physical components.

Using the cyclic lifecycle (CLC) Paradigm of the software lifecycle, we can modify it to implement an extreme component-base system. Fig.1 below shows CBD cycle compared with the cyclic life cycle model. It is important to observe that requirements identification and a design in CLC process is combined with finding and selecting components in component approach. Wolak (2001), pointed out that every software must undergo routine life cycle phases which may include requirements, design, installation, and commission, depending on the type of model under consideration. However, the component-based systems development process steps include:

1. Find components which may be used in the system: This phase (step) is an issue of both technology and business and hence requires that a vast number of candidates (people) and tools must be on ground for this task. The possible components are found and listed here for further investigation.
2. Select the components which meet the requirements of the system: fulfilling completely component requirements is impossible and hence a trade-off analysis is carried out here. Most times, the system architecture is adjusted and the requirements are reformulated to accommodate the existing components.
3. Alternatively, create a proprietary component to be used in the system: This requires more human efforts and lead-time and hence less attractive. Core-functionality components, however, are developed internally in order to provide the competitive advantage of the product.
4. Adapt the selected components so that they suit the existing component model or requirement specification: it is important to notice here that some components can be directly integrated into the system, while others may need modification through parameterization process or code wrapping for adaptation etc.
5. Compose and deploy the components using a framework for components: usually functionality is provided by component models.
6. Replace earlier with later versions of components: This step corresponds with system maintenance. Bugs may have been eliminated or new functionality added.

Software architecture and component-based development

Software architecture and components are closely related. Bass, Clement and Kazman (1998), defined computer software architecture as the structure or structure of the system, which comprise software components, the externally visible properties of those components and the relationships among them. Component technologies focus on composition and deployment at execution time or near execution time. Thus, the architecture remains recognizable during the application or system execution in a component system. Component-based software engineering therefore, embraces the total lifecycles of components and component-based systems and all the procedures involved in such lifecycles.

Using top-down design approach, software design begins with determining its architecture and structuring the system in smaller parts, as independent as possible. The functionality-based architecture design is the first phase of the structuring, and then followed by the software architecture assessment. Defining the software architecture gives birth to the components selection or development. Different categories of components can then be distinguished in relation to the requirements of the system; viz: special purpose components (developed specifically for the system), reused components, (internally developed for multiple usage), and final commercial components (COTS). Finally, pre-existing components typically need to be integrated into the system using glue code or a modification of the components themselves.

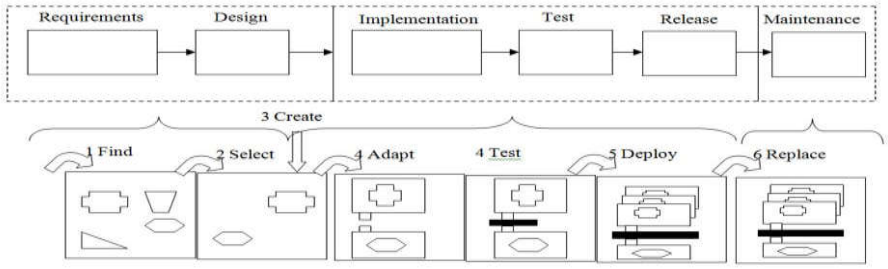

Fig. 1 The Component-Based Development Cycle Compared with The Cyclic Life Cycle Model

However, the top-down approach does not encourage the reuse of pre-existing components, since the pre-existing do not exactly fit into the system. However, the solution to this challenge is to use a mix of top-down and bottom up approaches. This will allow for the merging of common techniques, methods and tools. Architectural definition language (ADLS) such as ACME can be used for designing component-based systems and implemented for the existing

component models. The architectural analysis helps in making decision in component selection. In summary software architecture and component-based development are successfully used in the development of software product lines from which many variants of a product are delivered (ACME, 2001). The component-based approach and architectural design play important role in product configuration management (Larsson and Crnkovic, 1999).

Unified modeling language (UML) and component-based systems modeling: Arlow and Neustadt (2002) defined the unified modeling language (UML) as a graphical language for visualizing, specifying, constructing and documenting the artifacts of a software-intensive system.

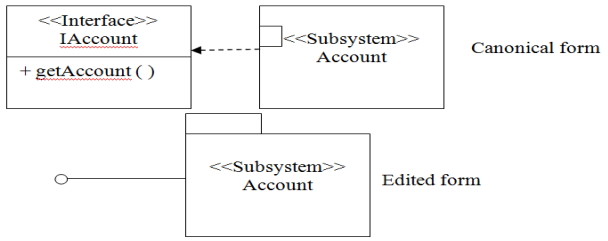

Fig. 2. UML Component

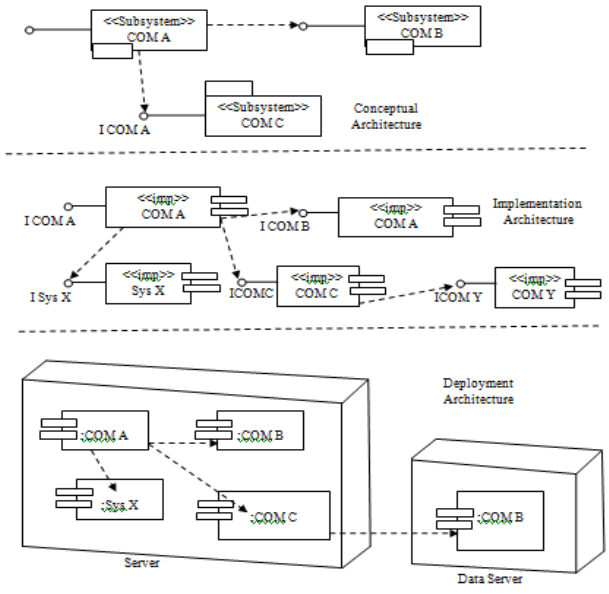

Fig.3, shows the three aspects of system architecture. The conceptual architecture is a result of a top-down system analysis and design. In at least the first step, the conceptual architecture is not different from a non-component-based design. In the conceptual part the components are expressed by UML packages with the <<subsystem>> stereotype. In the implementation architecture part, the physical components are represented by UML components and the <<imp>> stereotype. UML is not specialized for CBD and hence certain extensions to standard UML (e.g. naming convention or stereotypes) are required.

It is a language for specifying but not a method or procedure and offers a standard way to write systems blue prints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, data schemes, and reusable software components. UML can be used for both component and system modeling. Component-based design, concentrates on interface definitions and collaboration between the components through the interfaces. The design process continues with the modeling of the system with physical components, which do not necessarily match the logical structure. These may be pre-existing components, with interface already specified and possibly in need of wrappers. One logical component, identified in the first phase of design, may consist of several physical components. Finally, there is a deployment aspect, the components being executed on different computers in a distributed application. However, for non-component-based approach, first the design is important while direct mapping takes place, between the conceptual and implementation level and the deployment phase is the same for the whole application. In principle, UML can be utilized to provide support for designing component-based systems covering all these aspects. Interfaces are presented as multiple subsystems, which indicate the possibility of changing the implementation without replacing the interface. An interface can be presented in two ways: canonical form or edited form see Fig. 2 above.

Conclusion; The major problem encountered in component-Based system development is acquisition and elicitation of requirements in combination with COTS selection as this is a multi- criteria decision process. The solution to this great problem is the use of CLC paradigm which modifies this to implement an extreme component- Based system. The paper was able to critically compare the six steps involved in CLC and CBD cycle. The pre-existing components needs to be integrated into the system using glue code or modification of components themselves, but the top-down approach does not encourage the reuse of pre-existing components. The paper suggests a mixture of top-down and bottom-up approach as a solution to this problem. Software architecture and CBD are used in the development of a successful software product lines and especially in product configuration management. The author in this paper successfully presented UML as a standard way of writing systems blue prints such as business processes, system functions, programming language statements, data schemes, and reusable software components. The author presented UML as the best language for both component and system modeling and component-Based development for interface definition and collaboration between the components via the interfaces

REFERENCES

ACME, 2001. ACMC Architecture Definition Language, sourced: http://www.cs.cmu.edu/~qeme/

Arlow, J and Neustadt, 2002. UML And The Unified Process: Practical Object-Oriented Analysis And Design, Pearson Education Ltd, London.

Bass, L., Clements, P and Kazman, R. 1998. Software Architecture In Practice, Addison Wesley, London.

Crnkovic, I. 2001. Component-Based Software Engineering – New Challenges In Software Development; Software Focus, 2(14), Lund.

Larsson, M and Crnkovic, I. 1999. New Challenges For Configuration Management, Proceedings Of 9th Symposium On System Configuration Management, Lund University, Lund.

Osuagwu, Oliver E. 2005. Software Engineering: A Pragmatic And Technical Perspective, Oliverson Industrial Publishing House Owweri Nigeria.

Wolak, R.G. 2001. DISS 725 – System Development Research Paper 2, Software Requirements Engineering Best Practices, School Of Computer And Information Science, Southeastern University, Nova.