COMPONENT-ORIENTED PROGRAMMING: OBJECTED-ORIENTED AND BEYOND

Hieu D. Vu, Ph. D.
Hubei University of Economics
School of Information Management
No 8 Yangqiaohu Rd., Jiangxia
Wuhan, PRC
hieu.vu52@yahoo.com

Abstract
This paper addresses the following questions: What is software component? How to build component? What is component-oriented programming, and how the programming languages contribute in this new programming paradigm? Since the development of computer, scientists, computer experts, have been trying to find a better way to build software applications. Numerous programming languages were developed, and the newer one has more power, capabilities to facilitate programming. Along with the newer, object-oriented capability languages such as C++, Java, C#, programming paradigm is also changed, from the structured to object-oriented, and to component-oriented programming. This paper considers books, and research papers on similar topics that might provide a deeper understanding of software component, and the answers to these questions above.

Keywords
Computer component, Programming languages, Object-oriented, Software application, Programming paradigm
I. INTRODUCTION

The computer technology has changed rapidly. A personal computer we buy today is much more powerful than the one we bought just three years ago. As with the changes in computer hardware, and the demands of the industries, the way we write computer program, to build application software is also changed. Computer researchers, experts have been finding a better way, more efficiency and more effective. Frohlic classified three software development paradigms: structured, modular and object-oriented [1]. Component-oriented programming could be considered as a new paradigm in software engineering even it has been around more than ten years [2].

Example of such programming includes Sun Microsystem’s JavaBean, a reusable software component that can be manipulated in a builder tool. Some JavaBeans are GUI elements such as buttons, or check boxes. Others are more complicated components such as database viewer, data feeder [3].

In general, components are units of deployment. A software component is what is actually deployed. It is a separated, isolatable part of a software system in a component-based approach [4]. The unit of deployment is static such as a class, a set of classes, or a framework of classes that are bundled together into a package. A component can be regarded as a collection of one or more classes, but it is different from class.

II. SOFTWARE COMPONENT

In a component-based approach, components are similar to class. That is component define and create objects. Both components and classes implement functionality through the behavior descriptions called interfaces. However, unlike classes, components may be implemented by a single class, multiple classes, or non-object-oriented procedures. Moreover, component names, unlike class names may not be used as type names. In components, interfaces are types and are separate from implementations.

Ed Roman defines a software component is programming code that implements a set of well-defined interfaces. It is manageable, chunk of logics. Components are not entire application. It is just a piece of puzzle in an application [5]. Heineman and Councill define a software component is a software element that conforms to a component model and can be deployed independently without modification to a composition standard.

According to Brown (2000), a component is distinguished by three main characteristics:

- A component is an unit of deployment based on component model which defines the rules for component.
- A component provides a packet of one or objects implementation.
- A component is an unit of assembly for constructing a system. Each created using Object-Oriented Programming Language or other technology [6].

II. 1. COMPONENT MODELS

Software components must conform to the standards defined by a component model. A component model defines a set of standards for component implementations, naming, composition, and deployment [7]. There are several component models available today such as Object Management Group (OMG) CORBA Component Model (CCM), Microsoft’s COM/COM+ family, and Sun Microsystems’s JavaBeans and Enterprise JavaBeans (EJBs). For example, Microsoft’s Component Object Model (COM) requires each COM
component must have an IUnknown interface. Sun Microsystems defines some main characteristics of JavaBeans as following:

- If a bean has a property X, it should have public methods setX, and getX to assign and return value of the property.
- If a bean can generate events of the class YEvent, it should include public methods: void addYListener(YEvent), and void removeYListener(YEvent).

II. 2. ELEMENTS OF A COMPONENT

- Interfaces. One important concept about component is that it has a clearly defined of interfaces. An interface standard is required to enable software elements interact with one another. It defines what can be an interface. A component supports a provided interface if the component contains all implementations of operations defined by the interface.
- Composition standard. A component should encapsulate all necessary data and operations to perform its task independently and separated from the operating system. From the definition, a component is an independent unit for deployment.
- Component model implementation. Is a set of executable software elements necessary to support the execution of components in a component model

III. COMPONENT-ORIENTED PROGRAMMING.

The concept of software components has been around for a long time, but computer professionals understood “components” in their own different ways. A similar situation for “Objects and Object-Oriented Programming” until the late 1980s then things cleared out with the development of the most popular, fundamental object-oriented programming language C++. Now object-oriented programming is widely understanding and accepted, and a new programming paradigm has begun to emerge. It is component-oriented programming, a combination of modular and object-oriented programming mechanisms.

There are many debates concerning what exactly are and are not components. Some experts say components are coherent packages of useful behaviors. Others concentrate on components as physical, deployable unit software. Regardless of the differences, the principle concept of Component-Based Development (CBD) is to build systems from well-defined, independent units. Many professionals in the software industry are beginning to see the Component-Based Development (CBD) is an interesting approach to application development. This approach promises the reducing cycle time for the development and improving the quality of software.

III. 1.SOFTWARE REUSABILITY.

Why object-oriented and component-oriented programming are widely accepted and gain popularity quickly? The key answer for this question is “Software reusability”. Both object and component-oriented programming have the software reusability advantage and benefit. Reusability is a very important concept in software engineering. It saves time, energy for programmers, developers in building applications and of cause it also saves cost for the firm in the software development process. Instead of building the application from the scratch, we can use existing, available software components and assemble them to form a software system, or an application. These software components are available from software vendor that included with the description of the quality, functionality, and system requirement of the components. The software component
vendor must be well knowledgeable and has expertise in the component development to ensure the quality of the software products.

IV. DEVELOPMENT OF SOFTWARE COMPONENTS.

For many years, software developers have been trying to build modular and reusable software components by developing models for these components, the way the components should be constructed so they can be connected and work with each other. The goal for application developers is to make integration of software components a reality, so components can be reuse, assemble to make application. How to build software components? We need to design the software components in an implementation-neutral fashion This level of software design is independent of programming languages, even the latest language, and programmer prejudices.

In developing software components, several rules must be followed. Architectural conventions and rule must be explicit, interfaces be specified clearly unambiguously. Typically, components will be assembled by users, persons who are not developers, and potentially long after they were built. Therefore the relationships between implementations, and interface specifications, and user requirements must be verifiable, testable in a systematic way.

The development of a successful component is a complicated task. Heineman and Council believe that we can increase the chance of success by careful designing the Software Component Infrastructure, and each individual component. The design starts with clear objective, which states the performance, and the result in the design of a set of software components that will form the infrastructure of the final software system. Paul Allen gives a more detail how to develop components. The Component-Based Development (CBD) involves a wide range of provisioning strategies of which the development is only one part. It should emphasis on the market awareness of available components. The range of the CBD should across a full Software Development Life Cycle (SDLC) and also the reuse of component knowledge and models. According to Allen, the CBD involves three basic ideas:

1. Planning, analyzing, and designing. This is an abstract level in term of interfaces that offers adaptive business capacity to business processes.
2. Provisioning. Using implementation techniques that include the reuse of available software, outsourcing, and extending components bought in market to create new one.
3. Deployment. Integrate the component into an infrastructure that supports the executing, managing, and upgrading the solution as a set.

To facilitate the process of development of components, there should be a standardized way for building, manage, and maintain components. Roman states that this approach should include the following:

1. Tools for developing components. The process of building component should allow the developer to focus on coding the core logic behind the components. This will promote the rapid application development (RAD) technique. For instance, an Integrated Development Environment (IDE) such as the Symantec’s Visual Café, IBM’s VisualAge for Javaassists Java developers building, debugging components rapidly.
2. A container to manage deployed components. This component container provides runtime environment. It also provides a set of frequently used services for components. For example, the container could automatically instantiate new components as necessary, to relieve this burden for the developers.

3. Tools for deploying, maintaining components. This should come along with the components, when they are purchased from components vendors. For example, a company that buys a component should be able to customize it for a particular environment. The company should also be able to maintain the component for which it purchased.

IV. 1. PROGRAMMING MODELS.

Traditional programming model is primarily based on the composition of functions or procedures that are coded within a program or from a library. In this context, programming style is caller driven. According to Szypersky, interfaces list call points or procedures or functions. This does not change when moving from traditional to object-oriented programming. As long as the objects are passive, programming style still remains caller driven even object-oriented is considered a higher programming technique.

The programming model changes when emphasis in connections between objects rather that at callers. A connection is really a binding between a caller and a callee, which are the two ends of a connection. Interface in this setting needs to describe both caller and calling point. Microsoft introduced Connectable Objects approach to speak of incoming and outgoing interfaces. Incoming interfaces are the traditional one, lists all points (methods). Outgoing interfaces declare what operations a component could invoke if it is properly connected.

How can we ensure that two people understand the same meaning of a flow chart? The Unified Modeling Language (UML) provides a standard language that can be used in many different ways. It can be divided into three distinct models depending the semantic levels:

a. Conceptual models. Software that identifies the concepts in the domain being study.

b. Specification models. Software that defines the specifications of the component. It models the “outside” of the component, not the “inside” implementation.

c. Implementation models. Models detail the implementation design of the software, the “inside” of the component.

Heineman and Councill have a similar use of Unified Modeling Language (UML). UML can be used to model a component’s design and implementation. It is also useful in documentation of reuse feature. A component has a logical (abstract) and a physical (implementation) part, and both needed to be considered in defining the component.

The logical representation of a component is modeled using UML’s subsystem, which can be viewed as a subtype of UML package. Therefore, it can realize the interfaces that are the operations in subsystem specification. An interface can be realized by multiple subsystems, and a subsystem can also realize multiple interfaces. Any subsystems that realize the same interface can be used to substitute one another. This is a major advantage “plug and play” of a component-based system.

The implementation representation of a component defines how its logical representation is implemented in a chosen environment (language, model). The implementation must include any implementation dependencies between components.
IV. 2. PROGRAMMING LANGUAGES FOR COMPONENT-ORIENTED.

Harvey M. Deitel remembers the frustration in the 1960s by software developers when they tried to develop a large-scale project. In summer 1967, a company “decommitted” from producing a commercial product that involved hundred of programmers for several years. It is difficult to get the software right. Software development is a complex process. Just as object-oriented programming (OOP), the component-oriented programming (COP) is defined in a typical object-oriented programming. Component-oriented programming requires the support of:

- Polymorphism (substitutability, replaceable).
- Modular encapsulation (higher level of information hiding).
- Dynamic late binding and loading (independent deployment).
- Type (class) and module safety.

What programming languages, which are capable, and suitable for creating components? We have been trying to develop a modern language that increases the power of each line of code written (statements). It is clear that high-level languages such as Fortran, Cobol, or C have more power than the low-level Assembly language.

Object-oriented programming languages (OOPLs) have become popular because they allow software engineers to develop units of software called classes that maps directly to the real world of entities in a particular domain. The OOPLs are well suited for developing business components, but great care must be taken when building large-scale systems.

Any languages with object-oriented capability can be used to build software components, since component-oriented based heavily in object-oriented programming. Some author such as Ed Roman prefers Java “Java: An Ideal Language for Component Architectures”. Java supports the separation between interface and implementation. By separation, we can replace an implementation by a more effective one (plug-and-play feature).

Examples of Java components are JavaBeans, Enterprise JavaBeans (EJBs). JavaBean components regarded as small-grained application bits used in assembling larger components or an application. They are development components not deployable components. The EJB is defined as a deployable component called enterprise bean. They are larger and ready to be deployed. Other two types of Java components are Applets that can be deployed in a Web page, and Servlets deployed in a Web server.

Other authors Tom Archer and Andrew Whitechapel favor the .NET platform. The breadth and depth of the Microsoft .NET technologies delivers is astounding. C# is a new language, is also the first component-oriented language in the family of C, C++ programming languages. C# is simple, object-oriented and type-safe programming language that derived from C, and C++. It is a combination of Visual Basic and C++.

C, C++ programmers should feel comfortable with C# since it borrows core features such as expressions, statements, and overall syntax from C, and C++. Other than Java, C# allows direct pointer and address manipulation. The C# type system is also unified everything in C# is an object. It bridges the gap between value and reference types, allowing any pieces of data to be treated as objects.
V. CONCLUSION.

The major advantage of component-oriented programming over other programming paradigms is that its components promote rapid application development. In Object-Oriented Programming paradigm, the developers still need to write the infrastructure for the application. In component-based approach, the developers only need to assemble the components together. As a result, IT professionals can build applications quickly by assembling prewritten components rather than writing the entire application from scratch.

“Object orientation has failed but component software is succeeding”. First, by definition, an object technology encapsulates state and behaviors (data and functions), allows inheritance and polymorphism. The definition does not mention independence, and compositions therefore object technology can be used to build single application. Second, object technology ignores the economic and marketing aspect. We have a small number of vendors of classes such as the classic Microsoft Foundation Classes (MFC). The MFC serves as a tool to simplify and unify programming in Microsoft’s operating system environment, and applications.

Recently, based on true component technology, more successful vendors began to market their products. ComponentSource.com or Flashline.com sold thousands of ready-made software components mostly in the COM and Java categories, and VCL components from Borland International (Builder of C++ and Delphi), and Microsoft’s products .NET components. Other software companies such as ILOG and Rogue Wave Software focus on C, C++, and Java components for simulation/optimization, visual presentation, and rule-based applications. ILOG approached $80 million in its fiscal year in 2001.

Microsoft Corporation also released their new component products: COM+ as an extension of the Component Object Model (COM). COM+ builds on COM integrated features, making it easier for developers to create and use software components in any programming languages. Distributed Component Object Model (DCOM) is a protocol for components to communicate over a network in a reliable, secure environment [8].

In a recent report, Gartner Group predicted that by 2003, up to 70% of all new software solutions will be created with “building block” such as components and templates. A survey of worldwide markets and trends by International Data Corporation (IDC) estimated that by 2003, the worldwide market for components building and assembling would exceed $8 billion with $2 million spent on acquiring components.
References:


