FRAMEWORKS BETWEEN COMPONENTS AND OBJECTS

Mohamed Belal[1], Ayman Khedr[2], Ahmed Gohar[3]

[1]Prof. Computer Science Department, Faculty of Information and Computer (Egypt)
dr.mohamedbelal@gmail.com
[2]Dr. Information System Department, Faculty of Information and Computer (Egypt)
aymankhedr@gmail.com
[3]Information System Department, Faculty of Information and Computer (Egypt)
mcpgohar@hotmail.com

ABSTRACT

Before the emergence of Component-Based Frameworks, similar issues have been addressed by other software development paradigms including e.g. Object-Oriented Programming (OOP), Component-Based Development (CBD), and Object-Oriented Framework. In this study, these approaches especially object-oriented Frameworks are compared to Component-Based Frameworks and their relationship are discussed. Different software reuse methods impacts on architectural patterns and support for application extensions and versioning. It is concluded that many of the mechanisms provided by Component-Based Framework can be enabled by software elements at the lower level. The main contribution of Component-Based Framework is the focus on Component development. All of them can be built on each other in layered manner by adopting suitable design patterns. Still some things such as which method to develop and upgrade existing application to other approach.

KEYWORDS

Component-Based Development, CBD, Component-Based Framework, CBF, Framework, Object Oriented Framework, OOF, Object-Oriented Programming, OOP.

1. INTRODUCTION

Significant improvements in software productivity and quality reducing development costs provided by software reuse. One of the main development approaches for business and commercial systems is reuse-based software engineering. Ranging from fine-grain functions to entire application systems[1]. It is difficult to reuse medium-grain program components, as it is significantly larger than individual objects or procedures, with more functionality, but they are smaller and more specific than application systems[1].

Since then most influential OO-languages in industry have been C++ and Java. As a result of research and development (R/D) in OO many tools and technologies have been introduced to support it as modelling languages, application servers, OO-database or relational mapping tools and OO-based development processes. Then Component-based software engineering (CBSE) emerged as a reuse-based approach to software systems development[1]. The basic idea is simple: "When developing new systems use components that are already developed" [2]. Components are built to be used and reused in many applications. A component must be well specified, sufficiently general, easy to understand and adapt, easy to deliver and deploy and easy to replace[3].

In contrast to earlier object-oriented reuse techniques based on class libraries, framework is targeted for particular business units and application domains. Frameworks like MacApp play an increasingly important role in contemporary software development [4, 5].

Frameworks are one of the most appraised paradigms in software development. However, many of its claimed benefits have been addressed by previous approaches in software development. Additionally, software systems are not built using components as the only element of construction but lower level artefacts are also required[6] while developments in standardization promoted by major software vendors now mean that components can interoperate within a framework such as CORBA[1].

The research is structured as follows: first section 2, focus on the background of OOP and CBD paradigms are briefly introduced and the approach of software reuse discussed. Next, in section 3, discussing frameworks as reusable design. Then Next, in section 4, the both approaches are compared with each another from wither theoretical and practical prospective. Finally, in section 5, the findings of these comparisons are discussed and the concluding remarks are made.

2. BACKGROUND

The resulting increase in reuse should dramatically improve time to market, lifecycle cost, and quality[7]. It becomes apparent that what is needed is something that addresses to build individual systems that can be treated as atomic units and can easily be made to cooperate with each other[8].

Reuse of existing assets will enable projects to decrease the cost of developing and marinating of software. This software that has been multiple times will possess fewer defects than freshly coded components which results in decrease risk in creating new software when available reusable components already encompass the desired functionality and have standard interfaces to facilitate integration. Standard interfaces and common use of components across products facilitate case of use and interoperability[9]:.

Using reuse leads to improve functionality and performance, which can be amortized over multiple uses of the assets which is economically justified than the case where they would only be for single product. Design time is drastically reduced because key architectural decision have been made are embodied in the component model and framework[10]. Reuse-based software engineering becomes the main development approach for business and commercial systems ranging from fine-grain functions to entire application systems[1].

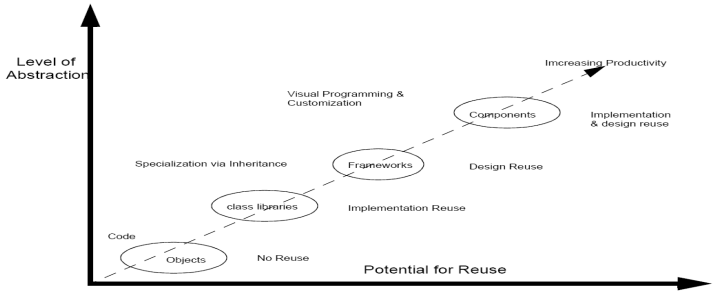

Figure1. Reusability through development techniques[9]

2.1. Object Oriented Programming and Development

In object oriented languages the execution flow of programs is passing messages between objects which represents concepts of the problem domain. Objects encapsulate related data and operations in one unit [11] this grouping of objects to classes and allowing subclasses to inherit and/or implement from their parent classes allows development using an existing code[12].

Object oriented modelling is usually the first step in object oriented development model. From high level view of the problem domain and use cases, the design phase continues to lower level implementation view of classes and objects. Modelling of these aspects is usually carried out in standard modelling language such as UML. Architectures oriented towards model driven architecture(MDA), where actual program code can be compiled from modelling effort[13]. Regarding OOSD, it increases code reuse, easy building on existing code, better change tolerance and decrease in errors by encapsulation.

2.2. Component Based Development

A software component has been described as ”a nontrivial, nearly independent and replaceable part of a system that full-fills a clear function in the context of a well-defined architecture” by Brown and Wallnau [14]. Clemens Szyperski and David Messerschmitt present the following five principles that a software component should have; Multiple-use, Non-context-specific, Compassable with other components, Encapsulated, A unit of independent deployment and versioning[15]

Unlike objects in the Object Oriented Programming (OOP), a component is an ” a widely adopted definition due to Szyperski [16] is the following: “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”.

Making applications from software components had been a dream in software engineering community since its very early time[17]. The purpose of CBSE is to improve quality of service (QoS), productivity and time-to-market in software development. Components might change to be updated based on the requirements of system changes. Summarily, they must be qualified and adapted if reusable components are available for potential integration [17].

3. FRAMEWORKS

Reuse approach is expected to revolutionize the booth development and maintenance of software systems. The resulting increase in reuse should dramatically improve time. Designers often trade simplicity for power [18].Frameworks represent code reuse. It is applied in more contexts and in the development process, so can have a larger impact on a project. But most design reuse is informal, and happens through using experienced developers. There is no standard design notation and there are no standard catalogues of designs to reuse so a single company can standardize, but this will not lead to industry-wide reuse [19].

Framework is a form of design reuse which is similar to other techniques for reusing high-level design, such as templates or schemas [20] . The main difference is that frameworks are expressed in a programming language, but others usually depend on a special purpose design notation and require special software tools.

According to the nature of the frameworks four major types had been classified as follow; Object-Oriented Framework which Consists of a set of classes that work together to solve a family of related problems. Software developers use inheritance and delegation to extend the framework. Examples are Model-View-Controller MVC, Microsoft Foundation Classes MFC[21]. Component-Based Framework which defines a set of Abstract Interactions which define the protocols by which components cooperate with each component takes on roles in various Abstract interactions which defined by a Component Framework that provides a conceptual framework in which to think about solutions to the problem domain addressed by the framework[22]. Enterprise or Business Framework which Provides a domain-specific, business solution that can be extended into an organization; not necessarily limited to Component-Based

Development; Examples: SAP R3, Siebold, Baan Company's BaanSeries [23].finally Technology Framework which Provides a standard, generic software foundation; not necessarily limited to Component-Based Development; Examples are COM, CORBA, Java.

4. COMPARING APPROACHES

This study will compare between object-oriented frameworks and component-based frameworks from both theoretical and practical prospective as follows:

4.1. Theoretical prospective

4.1.1. Comparing Based on Nature Dimension

The first important issue to understand the differences between these two approaches is to keep the sense of what differentiates the object and components. CBSE embodies "the buy, don't build philosophy". The main difference between a component and an object is that: a component is meant to be a runtime entity, whereas an object is an instance of a class [24]. OO with its deterministic and limited features (e.g., inheritance) cannot provide adequate flexibility required for building today's ever increasing complex software systems [25].

Inheritance [24] is a less useful concept in a component context that it is in an object-oriented context. The movement from inheritance based solutions to object compositions and message for forwarding and delegation, which was already on its way in the Object-Oriented (OO) world, has gained speed in the component world. A component comes to life through objects and therefore could contain one or more classes or immutable prototype. If only objects become visible to clients, there is no way to tell whether or not a component is purely object-oriented inside [26].

However, components have borrowed various concepts from objects with some additions and some exclusion. The most important additions are components' implemented nature and integration capabilities, even at runtime. A components' interface has more power that its counterpart does in an object because in addition to properties and methods [27]. Object are usually not thread-secure [24], because the designer knows or think he knows how the objects are going to be used. In a component context, he cannot be sure that, and therefore the components have to be secured.

4.1.2. Comparing Based on Enterprise Application Development Dimension

Component-based framework aims to realize long waited software reuse by changing both software architecture and software process. Because of the extensive uses of components, CBSE process is quite different from that of the traditional waterfall approach. CBSE requires focus on system specification and development, and also requires additional consideration for overall system context, [28]. Building complex objects can also use inheritance, while it is limited to composition in component[27].

4.1.3. Comparing Based on Reuse Dimension

Booth frameworks do work at different levels of abstraction, OO at object level, CBD at component level. This means that also reuse techniques are also working on different levels of abstraction. Software reuse can be divided to four different dimensions: abstraction, selection, integration and specialization [29]. To analyze differences in reuse techniques, these techniques can be categorized along dimensions as discussed in Table 1.

Table 1. Dimensions of reuse

Method/ Dimension	OOD	CBD
Abstraction	Classes, Interfaces, Data hiding and encapsulation	Component Interfaces, Encapsulation
Selection	Class libraries	Component libraries
Integration	Method calls, ORBs etc.	Function class, ORBs etc.
Specialization	Inheritance	NA

As it can be seen, CBD methods for reuse are quite different to methods. Discoverability and use of component repositories is basic CBD principles to enable reuse. For CBD and OO comparable method can be use of class and component libraries.

4.1.4. Comparing Based on Layering and Architectural Pattern Dimension

Layering is one key architectural principle in traditional software development. Martin Fowler[30] suggests three key architectural layers that are used in enterprise applications (Table 2). Note: These layers are logical layers in application and can be distributed differently in normal N-tier architecture between tiers. For example in case of a thin client, most of the presentation logic could be on the server and in some cases to have faster response times some of the domain logic could be implemented on the client.

Table 2. Three Principal Layers

Layer	Responsibilities
Presentation	Provision of services, display of information
Domain	Logic that is the real point of the system.
Data Source	Communication with databases, messaging systems, transaction managers, other packages

When compared to N-tier architecture, in OO- or Component-based development objects or components can call each other without tiered layering approach. Application front end is part of the presentation layer; domain logic is implemented in Business Logic. According to Michael Stal [31] same patterns that have been found usable in J2EE applications are directly applicable to Frameworks.

CBD with COTS Modern enterprise application systems developing process become more and more large-scaled, uneasily controlled, complex. Also, due to time-to-market, no developing standard pressure and growing demand of searching for a cost-effective, efficient and satisfying multiple Quality of service (QoS) requirement software developing paradigm, enterprise application are developed by using commercial-off-the-shelf (COTS) components rapidly. Comparison to the traditional approach in which software systems can only be implemented from scratch; these COTS components can be developed by different vendor using different languages and different computer platforms[32].

4.1.5. Comparing Based on How to Build Dimension

CBD is focused on the identification of reusable entities and relations between them, starting from the system requirements. The early design process includes two essential steps: Firstly, specification of system architecture in terms of functional components and their interaction, this giving a logical view of the systems and secondly, specification of system architecture consists of physical components.

Different lifecycle models, established in software engineering, can be used in CBD. These models will be modified to emphasize component-centric activities. Let us, consider, for

example, the waterfall model using an extreme component-based approach. Figure 2 shows the waterfall model and the meaning of the phases. Identifying requirements and a design in the waterfall process is combined with finding and selecting components[3]. The design includes the system architecture design and component identification/selection.

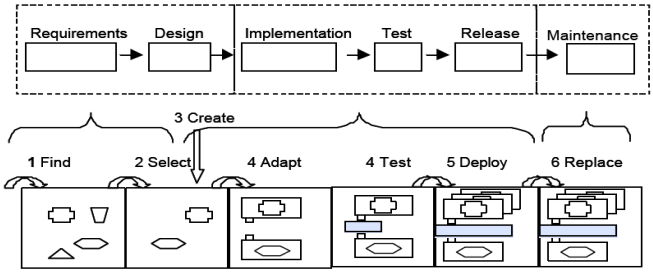

Figure 2. The development cycle compared with the waterfall model[3]

4.2. Practical prospective

4.2.1. Comparing Based on Extending Application Dimension

Regardless of the adopted approach, the developed software artefacts will change and evolve. Some changes can be made purely on the implementation level without changing the interface behaviour of objects or components. These kinds of changes do not necessarily pose problems in the software system because this does not change the way the element is used externally.

The version and evolution management mechanisms of different component-based technologies are discussed by Stuckenholz, 2005 [33]. In short, it can be said that most of the component-based technologies provide basic mechanisms for distinguishing between different versions of the component.

Extensions for component-based approaches depend on the used technology and their detailed discussion is out of the scope of this study [21]. The need for extensions depends on the adequateness of the original technology. Additionally, the layering of components and objects is meaningful here: for example, several security APIs exist for Java (objects) and they can be directly used in J2EE components. However, also the level of abstraction should be taken into account.

4.2.2. Comparing Based on Application Deployment Dimension

CBD changes the nature of software [8]. As illustrated in Figure 3, it reveals the need for redefinition of what an application is? Components become highly visible at run-time, and this affects the way software is built, assembled, deployed, tested, evolved, marketed, and sold. CBD is not only development approach nut also deployment approach, and this leads to a new way to market and buy software solutions.

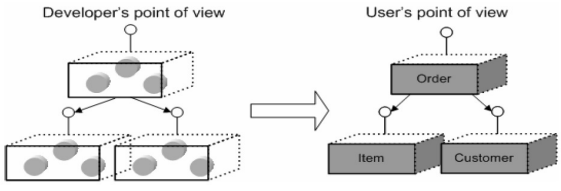

Figure 3. Component-based approach [8]

4.2.3. Comparing Based on Scope Dimension

It can also be stated that the scope and level of ambition associated with each of the approaches is different. To truly leverage from Components, one should aim at high component reusability at the enterprise level. Objects, on the other hand, can be adopted for a single application development project but they can still provide value. An enterprise-wide OO-architecture is of course also possible and significant, even if it would not result in the use of components or services.

The objectives of the discussed software development approaches Object-Orientation and Component-based frameworks are similar to some extent. They can all be considered as ways to promote software reuse and methods for structuring software systems into artifacts that can be managed separately for each other. However, these approaches have different scopes and focuses and they can be considered to operate on different levels of abstraction.

Conceptually, the approaches define different software system characteristics. By definition, CB frameworks are software components nature. Clearly, a good framework captures the basic characteristics of a component. However, the characteristics of CB frameworks define in more detail the framework architecture that these specific components constitute.

4. CONCLUDING REMARKS

Some of the problems with frameworks have been described already. Because they are powerful and complex, they are difficult to learn. This means they require better documentation and longer training than other systems. They are hard to develop, therefore they cost more to develop and require better programmers than normal application development. These are some of the reasons frameworks are not used more widely.

Although reuse is valuable, it is not free companies that are going to take advantage of reuse must pay its price. One of the strengths of frameworks is that they are represented by reusability.

One of the problems with using a particular language is that it restricts frameworks to systems using that language. In general, different object-oriented programming languages don't work well together, so it is not cost-effective to build an application in one language with a framework written in another.

Current programming languages are good at describing the static interface of an object, but not its dynamic interface. Because frameworks are described with programming languages, it is hard for developers to learn the collaborative patterns of a framework by reading it. Instead, they depend on the documentation and talking to experts. Patterns are one approach to improving the documentation. Another approach is to describe the constraints and interactions between components formally, such as with contracts [34]. But since part of the strength of frameworks is the fact that the framework is expressed in code, it might be better to improve object-oriented languages so that they can express patterns of collaboration more clearly.

Component-Based Frameworks are a practical way to express reusable designs. They deserve the attention of both researchers and practitioners. Although we need better ways to express and develop frameworks, they have already shown themselves to be valuable. Table 3 summarizing the main differences between both component and object based frameworks.

Table 3. A summary comparison of CBD and OOD [8, 24-27]

Comparison Factors	CB Framework	OO Framework
Flexibility	More flexible in terms of hardware and software	Less flexible
Reliability	Thread safe and secure	Dependent on developers'
Reusability	Run-time	Development-time
Building strategy	Composition is major Inheritance is unnecessary	Composition and Inheritance are both used
Deployment	Independent parts of Software	Monolithic software application
Interoperability	Provides communication between different technologies on different platform	Development is restricted with one or more technologies on one platform

REFERENCES

[1] I. Sommerville, Software Engineering, Publisher, City, 2010.
[2] I. Crnkovic, M. Larsson, Component-based Software Engineering-New Paradigm of Software Development, Publisher, City, 2001.
[3] I. Crnkovic, Component-based software engineering-new challenges in software development, in, IEEE, 2003, pp. 9-18.
[4] M. Fayad, D.C. Schmidt, Object-oriented application frameworks, Publisher, City, 1997.
[5] H.S. Chae, J.F. Cui, J.W. Park, J.G. Park, W.J. Lee, An object-oriented framework approach to flexible availability management for developing distributed applications, Publisher, City, 2009.
[6] M. Dragone, D. Lillis, R. Collier, G.M.P. O'Hare, SoSAA: a framework for integrating components & agents, in, ACM, 2009, pp. 722-728.
[7] C. Atkinson, J. Bayer, O. Laitenberger, J. Zettel, Component-based software engineering: The kobra approach, in, 2000.
[8] P. Herzum, O. Sims, Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise, John Wiley & Sons, Inc. New York, NY, USA, 2000.
[9] K. Ayse, Component-Oriented Modelling Of Land Registry And Cadastre Information System Using COSEML, in: Computer Engineering Department, Indian Institute of Technology, Bombay, India, 2002.
[10] F. Bachman, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, C.-M.U.P.P.S.E. INST, Technical Concepts of Component-Based Software Engineering, Volume 2, in, 2000.
[11] R.W. Sebesta, Concepts of programming languages, Addison-Wesley, 2002.
[12] Y. Zhang, R. Liang, Y. Zheng, M. Berry, Y. Wang, Y. Li, Teaching Object Oriented Database with Db4o, Publisher, City, 2012.
[13] T.O. Meservy, K.D. Fenstermacher, Transforming software development: an MDA road map, Publisher, City, 2005.
[14] L. Grunske, Early quality prediction of component-based systems–A generic framework, Publisher, City, 2007.

[15] Wikipedia, Software componentry, in, 2007.
[16] C. Szyperski, D. Gruntz, S. Murer, Component software: beyond object-oriented programming., in, Addison-Wesley, 2002.
[17] F. Siddiqui, Component Based Software Engineering: A Look at Reusable Software Components, Publisher, City, 2003.
[18] R.E. Johnson, Frameworks=(components+ patterns), Publisher, City, 1997.
[19] D.C. Schmidt, F. Buschmann, Patterns, frameworks, and middleware: their synergistic relationships, in, IEEE, 2003, pp. 694-704.
[20] M.E. Markiewicz, C.J.P. de Lucena, Object oriented framework development, Publisher, City, 2001.
[21] J.M. Lucassen, S.H. Maes, MVC (Model-View-Controller) based multi-modal authoring tool and development environment, in, Google Patents, 2011.
[22] G.T. Heineman, W.T. Councill, Component-based software engineering : putting the pieces together, Addison-Wesley, Boston, 2001.
[23] M. Wang, Integrating SAP to Information Systems Curriculum: Design and Delivery, Publisher, City, 2011.
[24] M. Huizing, Component Based Development, Publisher, City, 1999.
[25] M. Kavianpour, The Need for Component-based Software: Application of OMG CORBA in Building an Engineering Environment for Component Construction and Composition, Publisher, City, 1997.
[26] C. Szyperski, Components and objects together, Publisher, City, 1999.
[27] A. Dogru, M. Tanik, A process model for component-oriented software engineering, Publisher, City, 2003.
[28] I. Kaur, P.S. Sandhu, H. Singh, V. Saini, Analytical Study of Component Based Software Engineering, Publisher, City, 2009.
[29] C.W. Krueger, Software reuse, Publisher, City, 1992.
[30] M. Fowler, Patterns of enterprise application architecture, Addison-Wesley Professional, 2003.
[31] M. Stal, Using architectural patterns and blueprints for service-oriented architecture, Publisher, City, 2006.
[32] A. Gokhale, D.C. Schmidt, B. Natarajan, N. Wang, Applying model-integrated computing to component middleware and enterprise applications, Publisher, City, 2002.
[33] A. Stuckenholz, Component evolution and versioning state of the art, Publisher, City, 2005.
[34] Y. Jia, M. Tan, Y. Gu, The evolutionary approach to semantics-driven CBD automation, in, IEEE, 2002, pp. 98-104.