# Characterization and Architecture of Component Based Models

Er.Iqbaldeep kaur *(Author)*
Assistant Professor ,Department of Computer Science and Engineering, Rayat and Bahra Institute of Engineering and Bio-Technology ,Kharar, India
(eriqbaldeepkaur@gmail.com)

Dr.P.K.Suri
Dean and Professor, Computer Science and Application Department, Kurukshetra University, Kurukshetra, India
( pksuritf25@yahoo.com)

Er.Amit Verma
Assistant Professor , Department of Electronics and Communication Engineering, Rayat and Bahra Institute of Engineering and Bio-Technology,Kharar,India
(ervermaamit@gmail.com)

***Abstract*—Component based Software Engineering is the most common term nowadays in the field of software development. The CBSE approach is actually based on the principle of 'Select and Use' rather than 'Design and Test' as in traditional software development methods. Since this trend of using and 'reusing' components is in its developing stage, there are many advantages and problems as well that occur while use of components. Here is presented a series of papers that cover various important and integral issues in the field concerned. This paper is an introductory research on the essential concepts, principles and steps that underlie the available commercialized models in CBD. This research work has a scope extending to Component retrieval in repositories and their management and implementing the results verification**.

***Keywords- Components, CBSD, CORBA, KOAYLA, EJB, Component retrieval, repositories etc.***

## I. Introduction

The advantages of component based development include lesser development time, lower costs, reusability and better modification. A component is the basic building block of an application or system created with CBD. Generally, a component can be defined as an independent and replaceable part of a system that fulfills a clear function. It works in the context of a well defined architecture and can communicate with other components through its interfaces (Fig. 1). Although the basic principle of 'Plug and play' is very promising, but it also brings in some practical difficulties faced by the stakeholders involved. For instance, when we buy a component, we do not know exactly about its maintenance, the security arrangements and the most important its behavior when integrated with other components. There exist some models in the market that, to an extent, provide us with some standards and interfaces to aid the intercommunication process of components within integration. The models enable the independently designed components to be deployed and ease the communication between them. Rightly stated, it can be said that a component model supports components by forcing them to conform to certain standards and allows instances of these components to cooperate with other components in this model (fig. 2). In the absence of component models, there would be obvious non-cooperation among independently developed components, so the aim of 'independent deployment and assembled integration' of components

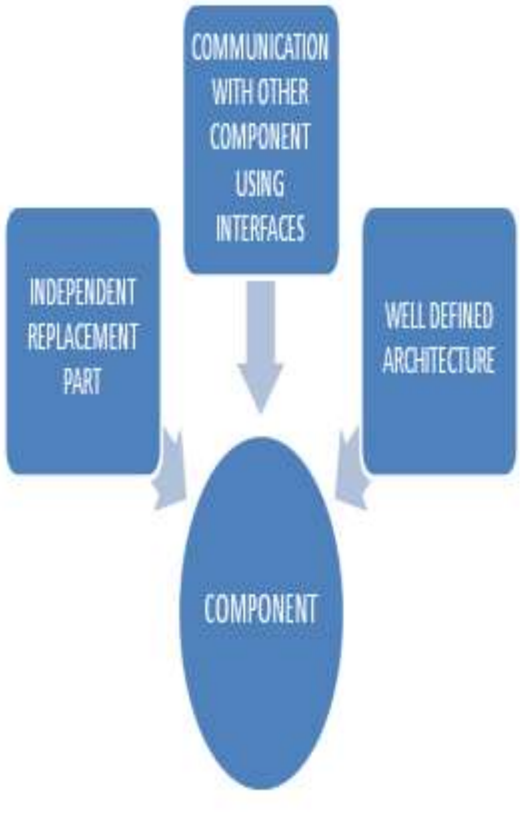

COMMUNICATION WITH OTHER COMPONENT USING INTERFACES
INDEPENDENT REPLACEMENT PART
WELL DEFINED ARCHITECTURE
COMPONENT

Fig. 1 Component and its features

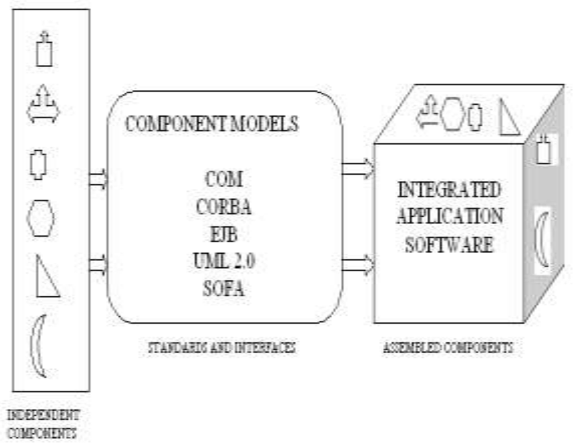


Fig. 2 Component Models provide interface to components

would not be realized. Thus, these models play a significant role in making the real goal[26-28] of CBD achieved. In the next sections, a detailed characteristic listing has been done for the main component models in market.

## II. Existing Component Model(Background & Related work)

The cornerstone of any CBD methodology is its underlying component model which defines what components are, how they can be constructed, how they can be assembled[1]. Component-based approach has shown considerable successes in recent years in many application domains like Distributed and web-based systems, desktop and graphical applications etc. In these domains the general-purpose component technologies, such as COM, .NET, EJB, J2EE are used [12]. According to [5], there are some commercial players involved in the software component revolution, such as BEA, Microsoft, IBM and Sun. [5] also states that among the component infrastructure technologies that have been developed, three have become somewhat standardized: OMG's CORBA, Microsoft's Component Object Model (COM) and Distributed COM (DCOM), and Sun's JavaBeans and Enterprise JavaBeans .

Most of the literature contains description about three major component models viz, OMG's CORBA, SUN's EJB and Microsoft's COM. The present work includes these three and some other less known models that are still maturing. At present there are various component models that are being used. These are shown pictorially in figure 3. Some approaches, such as Visual Basic Controls (VBX), ActiveX controls, class libraries, and JavaBeans, make it possible for their related languages, such as Visual Basic, C++, Java and the supporting tools to share and distribute application pieces. But all of these approaches rely on certain underlying services to provide the communication and coordination necessary for the application. The infrastructure of components, called a component model, in fact, acts as the "plumbing" that allows communication among components [9].

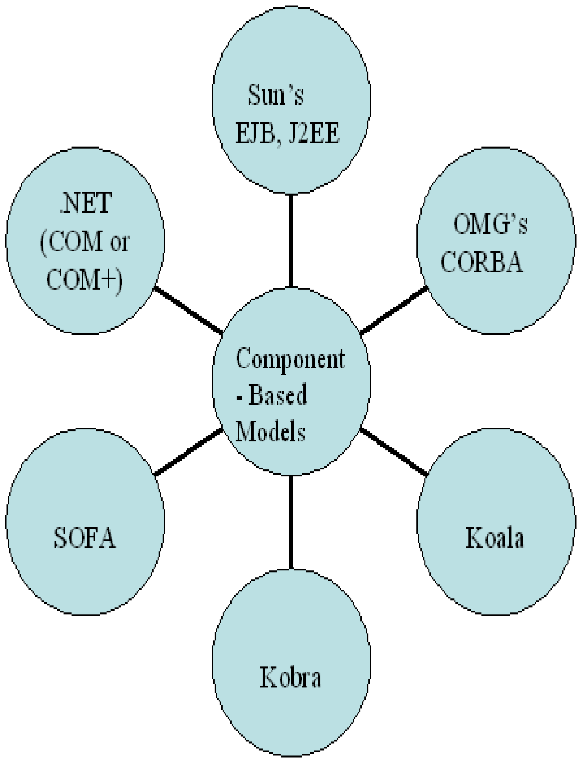


Fig.3 Component Models in Market

Generally Component Models work in three different service categories as follows: Basic, Distributed & Enterprise For example, the basic services include the simple component model version like COM, CORBA or EJB. Similarly, Distribution is provided[32-33] with a communication protocol that has been added to the basic component model.

## III. Component Object Model

It provides platform-dependent, based on Windows and Windows NT, and language-independent component based applications. COM defines how components and their clients interact. This interaction is defined such that the client and the component can connect without the need of any intermediate system component. Specially, COM provides a binary standard that components and their clients must follow to ensure dynamic interoperability. This enables on-line software update and cross-language software reuse [7].The following features characterize COM model:

- A model for designing components that have multiple interfaces with dynamic binding
- COM is an open standard, with main platform as Microsoft Windows
- Interfaces are the only means for components to expose themselves
- The interfaces are binary which provide the obvious ease to implement the component in multiple programming languages such as C++, Visual Basic and Java.

- A COM component can implement and expose multiple interfaces
- COM helps client to locate server components and desired interfaces by establishing connection between client and server.
- Interfaces [35-36] are defined as unchangeable units (A basic COM rule is that one cannot change an interface when it has been released), hence solving the interface versioning problem. Each time a new version of the interface is created a new interface will be added instead of changing the older version.

DCOM is the protocol that is used to make COM location transparent. A client talks to a proxy, which looks like the server and manages the real communication with the server. [3] has stated on DCOM , the extension of the Component Object Model (COM) as follows. Distributed COM (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner.

DCOM is designed for use across multiple network transports, including Internet protocols such as HTTP. When a client and its component reside on different machines, DCOM simply replaces the local inter-process communication with a network protocol. Neither the client nor the component is aware the changes of the physical connections. COM+ is an extension to COM with technologies that supports various additional services like transactions, directory service, load balancing and message queuing. COM+ is implemented to connect the clients to the business logic, through an Internet Information Server (IIS) or DCOM, as shown in figure 4.

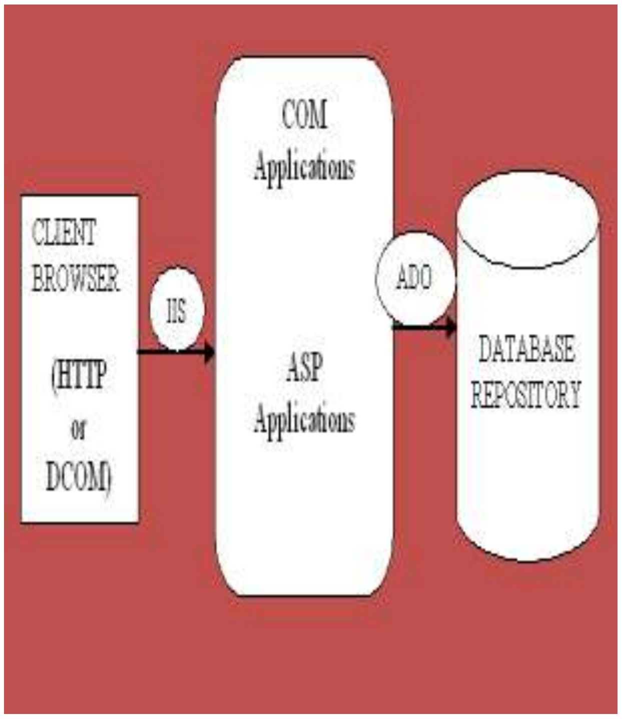


Fig.4 COM Architecture

The business logic uses ActiveX Data Objects (ADOs) to access the data in the databases.

## IV. ENTERPRISE JAVA BEANS(EJB)

U In accordance with [3], Java platform offers an efficient solution to the portability and security problems through the use of portable Java byte codes and the concept of trusted and non-trusted Java applets. Java provides a universal integration and enabling technology for enterprise application development, which includes:

- Interoperating across multivendor servers;
- Propagating transaction and security contexts;
- Servicing multilingual clients; and
- d)Supporting ActiveX via DCOM/CORBA bridges.

[8] has mentioned that the JavaBeans component architecture supports applications of multiple platforms, as well as reusable[24],[35], client-side and server-side components. JavaBeans and EJB extend all native strengths of Java including portability and security into the area of component-based development. The portability, security, and reliability of Java are well suited for developing robust server objects independent of operating systems, Web servers and database management servers. Sun's Java-based component model consists of two parts:

- JavaBeans for client-side component development
- Enterprise JavaBeans (EJB) for the server-side component development.

The following are the main features of EJB: EJB is part of the Java 2 Platform Enterprise Edition (J2EE) which includes remote method invocation (RMI), naming and directory interface (JNDI), database connectivity (JDBC), Server Pages (JSPs) and Messaging services (JMS). Fig. 5 shows the architectural style of EJB used in a three-tier application. EJB is designed so it can run together

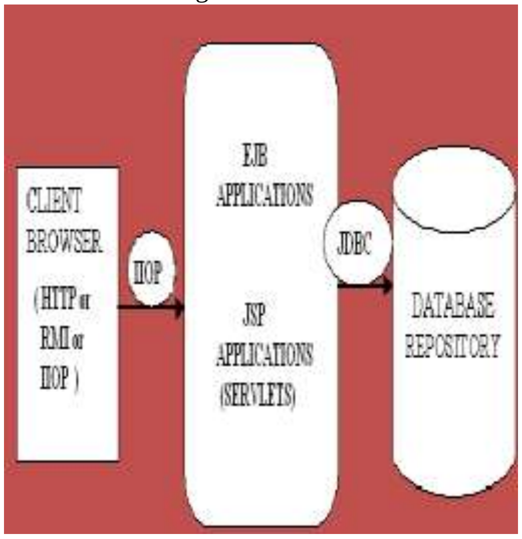

CLIENT BROWSER (HTTP or RMI or IIOP )
IIOP
EJB APPLICATIONS
JSP APPLICATIONS (SERVLETS)
JDBC
DATABASE REPOSITORY

Fig 5 EJB Architecture

with CORBA and access CORBA objects easily.

## V. Common Object Request Broker Architecure (CORBA)(Current Technology Used)

The Common Object[25],[29] Request Broker Architecture[30],[34] (CORBA) is a standard that has been developed by the Object Management Group (OMG) in early nineties. The OMG provides industry guidelines and object management[31]specifications to supply a common framework for integrating application development. Primary requirements for these specifications are reusability[21-24],[35] portability and interoperability of object based software components in a distributed environment. CORBA is part of the Object Management Architecture (OMA)[31] which covers object services, common facilities and definitions of terms. Object services include naming, persistency, events, transactions and relationships.

The following are the primary working principle of OMG's CORBA:

- The most important part of a CORBA system is the Object Request Broker (ORB).
- An object request broker (ORB) provides the basic mechanism for transparently
- Requests can be made through the ORB without regard to the service location or implementation.
- Objects publish their interfaces using the Interface Definition Language (IDL)
- Objects are stored in an interface repository where they can be found and activated on demand from the clients.
- The stubs and proxies are generated from the IDL specification

According to [3], CORBA manages details of component interoperability. Also CORBA is widely used in Object-Oriented distributed systems[6].

## VI. Less Popular Component Model Technologies Sofa Sofa 2 Koala Kobra

As stated by [13], the component model SOFA is a part of SOFA project (Software Appliances). It is a software system is described as a hierarchical composition of primitive and composite components. A component is an instance of a template, which is described by its frame and architecture. The frame is a "black-box" specification view of the component defining its provided and required interfaces. Primitive components are directly implemented by described software system they have a primitive architecture[37]. The architecture of a composed component is a "grey-box" implementation view, which defines first level of nesting in the component. It describes direct subcomponents and their interconnections via interfaces. The connections of the interfaces can be realized[38] via connectors, implicitly for simply connections or explicitly. Explicit connectors are described in a similar way as the components, by a frame and architecture. The connector frame is a set of roles, i.e. interfaces, which are compatible with interfaces of components.

SOFA 2 is a component system employing hierarchically composed components. It is a direct successor of the SOFA component model.

KOALA

Having most of its uses within Philips, Koala [14] offers explicit management of a special graphical notation that is very helpful in design discussions, and an elegant parameterization mechanism. Its partial evaluation techniques can calculate part of the configuration at compile time while generating code for that part that must be determined at runtime. In designing Koala, a strict separation is sought between component and configuration development.

- Koala components are units of design, development, and – more importantly – reuse.
- As in COM and Java, a Koala interface is a small set of semantically related functions.
- Koala components access all external functionality through requires interfaces which provides the architects with a clear view of the of the system's resources use.
- Koala components are designed independently of each other. They have interfaces to connect to other components, but this binding is late – at configuration time.

Koala has some extra features that are aimed at handling diversity efficiently: interface compatibility, function binding, partial evaluation, diversity interfaces, diversity spreadsheets, switches, optional interfaces, and Connected interfaces.

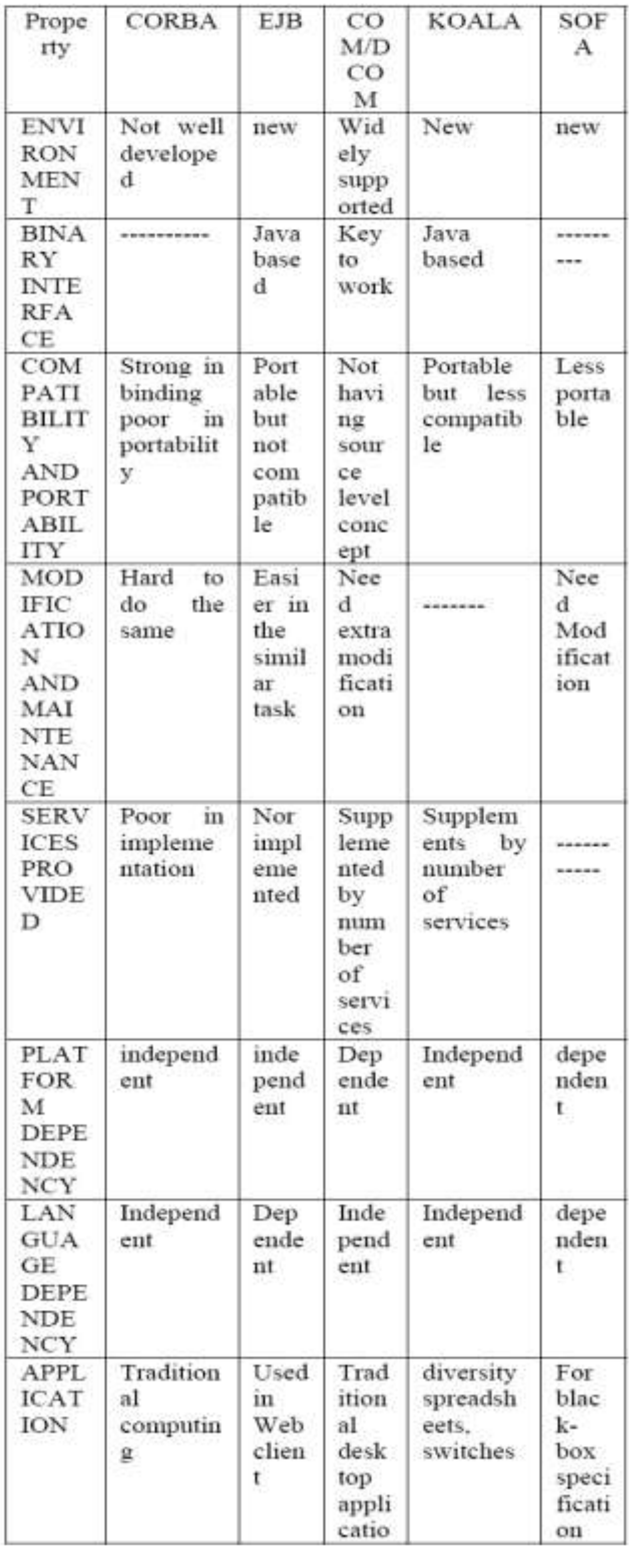
KOBRA

[16] states that KobrA is a UML-based method for describing components and component-based systems developed at the Fraunhofer Institute for Experimental Software Engineering at the beginning of the decade. The acronym stands for the term "Komponenten basierte Anwendungsentwicklung" – German for "Component-based Application Development". KobrA has been successfully used by a number of companies in industrial settings and has given rise to numerous specializations and offshoots . The original version of the method was developed for the UML 1.x flavor of the UML.

## VII. Conclusion and Comparison

Component-based systems result from adopting a component-based design with strategy, and software component technology includes the products and concepts that support this design strategy. By design strategy we mean something almost near to architectural style—a high-level design pattern and system described by the types of components in a system and their patterns of interaction [20]. Component based software development (CBSD) refers to the development of software component systems making considerable use of software components. Component based software development can help the software industry to realize productivity and quality gains similar to those achieved in hardware and manufacturing organizations. A detailed characterization of known component model technologies has been done in the present research work. The difference in all the model with respect to properties as shown in table 1 is illustrated. Some models like COM, CORBA and EJB are very well known among users and developers, whereas some other quite effective model technologies for component

Table 1: Comparative Study



| Property | CORBA | EJB | COM/DCOM | KOALA | SOFA |
|---|---|---|---|---|---|
| ENVIRONMENT | Not well developed | new | Widely supported | New | new |
| BINARY INTERFACE | ---------- | Java based | Key to work | Java based | --------- |
| COMPATIBILITY AND PORTABILITY | Strong in binding poor in portability | Portable but not compatible | Not having source level concept | Portable but less compatible | Less portable |
| MODIFICATION AND MAINTENANCE | Hard to do the same | Easier in the similar task | Need extra modification | ------- | Need Modification |
| SERVICES PROVIDED | Poor in implementation | Nor implemented | Supplemented by number of services | Supplements by number of services | ----------- |
| PLATFORM DEPENDENCY | independent | independent | Dependent | Independent | dependent |
| LANGUAGE DEPENDENCY | Independent | Dependent | Independent | Independent | dependent |
| APPLICATION | Traditional computing | Used in Web client | Traditional desktop applicatio | diversity spreadsheets, switches | For black-box specification |

Based software development are less popular as compared to these. Since the CBSE is a new discipline and is still maturing, a lot has to be done to find solutions to its associated problems which remain unsolved.

## REFERENCES

[1] Kung –Kiu and Zheng Wang, "A survey of Software Component Models", School of computer Science, University of Manchester, April, 2005, available at http://www.cs.man.ac.uk/preprints/index.htm

[2] A. Campbell. A Quality of Service Architecture. PhD Thesis, Lancaster University,1996

[3] Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes, Xia Cai, Michael R. Lyu, Kam-Fai Wong Roy Ko, The Chinese University of Hong Kong Hong Kong Productivity Council {xcai@cse, lyu@cse, kfwong@se}.cuhk.edu.hk roy@hkpc.org

[4] http://www.omg.org/corba/whatiscorba.html, Mar, 2000.

[5] W. Kozaczynski, G. Booch, "Component-Based Software Engineering," IEEE Software Volume: 155, Sept.-Oct. 1998, pp. 34–36.

[6] S.S.Yau, B. Xia, "Object-Oriented Distributed Component Software Development based on CORBA," Proceedings of COMPSAC'98. The Twenty-Second Annual International, 1998, pp. 246-251

[7] Y.M. Wang, O.P. Damani, W.J. Lee, "Reliability and Availability Issues in Distributed Component Ojbect Model (DCOM)," Fourth International Workshop on Community Networking Proceedings, 1997, pp. 59 –63.

[8] http://developer.java.sun.com/developer, Mar. 2000

[9] A.W.Brown, K.C. Wallnau, "The Current State of CBSE,"IEEE Software, Volume: 15 , Sept.-Oct. 1998, pp. 37- 46

[10] C.Szyperski, "Component Software: Beyond Object-Oriented Programming," Addison-Wesley, New York, 1998.

[11] G. Pour, "Enterprise JavaBeans, JavaBeans & XML Expanding the Possibilities for Web-Based Enterprise Application Development," Proceedings Technology of Object-Oriented Languages and Systems, 1999, TOOLS 31, pp.282-291.

[12] Component-based Development Process and Component Lifecycle by Ivica Crnkovic, Stig Larsson, Michel Chaudron

[13] 'Component Model with Support of Mobile Architectures' by Marek Rychllli, Brno University of Tcchnology, Czech Republic

[14] The Koala Component Model for Consumer Electronics Software, Rob van Ommering, Frank van der Linden, Jeff Kramer, Jeff Magee, IEEE Computer, March 2000, p78-85

[15] Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International Publication Date: 200, Pages 212 - 223 , Seattle, WA.

[16] 'Modeling Components and Component-Based Systems in KobrA' by Colin Atkinson

[17] Enterprise JavaBeans Specification. Version 2.0. Sun Microsystems. 2001.

[18] Object Management Group. The Common Object Request Broker: Architecture and Specification. version 3.0./02-06-33. 2002.

[19] F. E. Redmond III. DCOM: Microsoft Distributed Component Object Model. [sofa] Frantisek Plasil, Dusan Balek, and Radovan Janecek. SOFA/DCUP: Architecture for Component Trading and Dynamic Updating. Proceedings of ICCDS 98, May 4-6, 1998, Annapolis, Maryland, USA. IEEE CS Press. 1998.

[20] Bass, L; Clements, P.; & Kazman R. Software Architecture in Practice. Boston, Ma.: Addison Wesley, March 1998.

[21] zyperski, C, Component Software: Beyond Object-Oriented Programming, Addison Wesley, 1999.

[22] Cecilia Albert and Lisa Brownsword, Evolutionary Process for Integrating COTS-Based Systems (EPIC): An overview, Technical Report CMU/SEI-2002-TR-009 ESC-TR-2002-009, July, 2002.

[23] Jerry Zeyu Gao, Jacob Tsao, Ye Wu, Testing and Quality Assurance for Component Based Software, Artech House Publishers, 2003.

[24] David Garlan et al, "Architecural Mismatch: Why Reuse is so Hard", IEEE software, 1995.

[25] Ian graham, Object Oriented Methods, - Principles and practice, 3$^{rd}$ Edition, Addison Wesley, Object Technology Series.

[26] Ian Sommervilee, Software Engineering, 7th Edition, Pearson Education.

[27] R.S.Pressman, Software Engineering – A Practioners Approach, Fourth Edition, McGraw Hill International Series.

[28] Hafedh Mili et al, Reuse Based Software Engineering, Techniques, organization and Controls, John Wiley and Sons, 2002.

[29] Alencar, A. & Goguen, J. "OOZE," Stepney, S.; Barden, R.; & Cooper, D., ed. Object Orientation in Z, Workshops in Computing. Los Angeles, Ca.: Springer-Verlag, 1992.

[30] Allen, R.; Douence, R.; & Garlan, D. "Specifying Dynamism in Software Architectures," Proceedings of the 1st Workshop on Component-Based Systems. Zurich, Switzerland, 1997, in con-junction with European Software Engineering Conference (ESEC) and ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), 1997

[31] Baggiolini, V. & Harms, J. "Toward Automatic, Run-Time Fault Management for Component-Based Applications," Proceedings of the 2nd International Workshop on Component-Oriented Pro-gramming (WCOP97), in conjunction with the European Confer-ence on Object-Oriented Programming (ECOOP98, Brussels, Belgium, July 1998.

[32] Barnes, J. High Integrity Ada: the SPARK Approach. Boston, Ma.: Addison-Wesley, 1997

[33] Box, D. Essential COM. Boston, Ma.: Addison-Wesley, 1998.

[34] Ciancarani, P. & Cimato, S. "Specifying Component-Based Soft-ware Architectures," 60–70. Proceedings of the ESEC/FSE-Workshop on Foundations of Component-Based Systems (FoCBS), Zürich, Sep. 1997.

[35] Deline, R. "Avoiding Packaging Mismatch with Flexible Packaging," Proceedings of the 21st International Conference on Soft-ware Engineering. Los Angeles, Ca., May 1999.

[36] Della, C.; Cicalese, T.; & Rotenstreich, S. "Behavioral Specification of Distributed Software Component Interfaces." IEEE Computer (Jul. 1998): 46-53

[37] Dowson, M. "ISTAR and the Contractual Approach," 287-288. Proceedings of the 9th International Conference on Software En-gineering. Monterey, Ca, March 30-April 2, 1987. Washington DC, Baltimore, Md.: IEEE Computer Society and the Association for Computing Machinery, April 1987.

[38] Hissam, S & Carney, D. "Isolating Faults in Complex COTS-based Systems." Journal of Software Maintenance: Research and Practice, No. 11 (1999): 183-1999