ISSN:0975-9646

Component-Based Software Development with Component Technologies: An Overview

Jawwad Wasat Shareef[#1], Rajesh Kumar Pandey[*2]

[#]*Department of Mathematics and Computer Science, Rani Durgavati University, Jabalpur, M.P., India*

[*]*University Institute of Computer Science and Applications, Rani Durgavati University, Jabalpur, M.P., India*

***Abstract*— Component-based software development (CBSD) is an approach in which large software systems are built by assembling a set of previously developed software components that can be independently deployed, configured, adapted and connected together within appropriate software architecture. The benefits of this technology include, a shorter development time at a reduced cost with an increased degree of interoperability, portability and maintainability which gives a good prospect for this type of development.**

This paper presents an overview of CBSD; major activities involved in this process with an overview of most commonly used component technologies that are applied for component-based software development, along with their pros and cons. An attempt has been made to compare these technologies based on their functionality and mechanism, thus providing a roadmap to a developer in selecting the appropriate technology as per the requirements.

***Keywords*— Software component, Software Component Model, Component-Based Software Development, EJB, CORBA, .NET.**

I. INTRODUCTION

In Software Engineering the Component Based Software Engineering (CBSE) plays an important role, by "building systems from components", which is adopted from other reengineering fields, such as mechanical or electrical engineering. In context of CBSE comes Component-Based Development (CBD), which plays an important role in Software engineering. Main task of CBD is to build systems comprising of already built software units or components, thus promoting reuse concepts in CBD. This reuse concept reduces production cost as well as saves time by composing a system from prebuilt or existing components, instead of building them from scratch, these already prebuilt components can be reused in many systems.

The benefits that can be gained using CBD technology are increased reuse, reduced production cost and shorter time to market, which after much effort have been sought by software industry. To realize these benefits, it is necessary to have components that can be easily reused and using composition mechanism can be applied in a systematic way. As component based software engineering is based on the concept of component. We can find different definitions of component in a literature. An early and most commonly used definition proposed by Szyperski *et. al.*, [1] is as follows:

> "*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*".

The most important aspect of software component is reusing them in building large systems. When a component is deployed and executed after installation in different systems, it has to maintain its functionality. Components should be constructed in such a way that it must be possible to connect to a software component at run-time or dynamically; therefore a component is stated to be independently deployable. This approach maximizes the utilization of resources, thus giving the developer an option to use the components as and when required.

II. COMPONENT-BASED SOFTWARE DEVELOPMENT PROCESS

The term Component-Based Software Development (CBSD) is an appropriate and methodical approach, which involves the construction of an application by using prebuilt chunks, which were developed at different times, by different humans, and possibly with different concept and uses in mind [2]. Brown [3] stated four activities in component-based development approach:

- component qualification
- component adaptation
- assembling components
- system evolution

In the following subsections we discuss the above defined activities in some more detail.

A. Component qualification

It is a process of understanding whether a component which has been previously developed is suitable for reusing in target system. In a market place where number of prebuilt-components exists, then through this process components can

also be selected. The component qualification factors like standards based on reliability, predictability, usability, functionality, services that a component is providing should be considered.

B. Component adaptation

As the components are developed at different times, by different humans, and possibly with different concepts and requirements in mind. When these components are reused in a new system they must be adapted based on rules that ensure to minimize the conflicts between them. These conflicts can be avoided by applying adaptation technique called component wrapping [4]. Roger S. Pressman [5] defined wrapping approaches for component adoptions like:

- *White-box wrapping* examines the internal processing details of the component and makes code level modifications to remove any conflict.
- *Gray-box wrapping* is applied when the component library provides a component extension language or API that enables conflicts to be removed or masked.
- *Black-box wrapping* requires the introduction of pre and post processing at the component interface to remove or mask conflicts.

C. Assembling components

Assembling of components is done by using some well defined infrastructure, which helps in binding these components different from one another, thus forming a system. Component definition given by Heineman and Councill's [6] states that:

> "*A [component is a] software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard".*

which means a standard is needed to make the components work together. Major companies have defined standards for building components and some well known standards include Sun Microsystem's Javabeans and Enterprise JavaBeans (EJB), Microsoft's .NET and Object Management Group's (OMG) Corba Component Model (CCM) .

D. System evolution

System evolution plays an important role in order to meet business needs; it is like the process of replacement of old version or outdated components with new versions. To manage different versions of components and their impact on the applications which they are part of [7]. System evolution can be viewed as treating components in the form of plug-replaceable units.

III. COMPONENT-TECHNOLOGIES

There are many different component technologies widely used as per their applying requirements. Some of the most used technologies today are the Enterprise Java Beans (EJB) by Sun Microsystem's, CORBA by the Object Management Group (OMG), and .NET from Microsoft. These technologies will be described in detail in the following subsections.

A. *Enterprise Javabeans* (EJB)

Enterprise Javabeans (EJB) is a component developed in the Programming language Java. EJB is designed for inter-process components [8] as shown in Fig.1. Sun Microsystems' definition of Enterprise JavaBeans is:

> "*The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification*".

EJB's are server side software components [9] that can be deployed in a distributed multi tired environment. All EJB's must be hosted in a container (an EJB container can be a part of an application server) to be of any use. Furthermore, all EJB's must be written in the Java programming language. This gives high security and stability, and access to most features of Java. For instance, if a thread dies in a Java application, the application will still keep on running. There are no problems with pointers when using Java, which reduces the number of memory leaks. Java has an extensive and well tested library, and it is also platform independent. An EJB can compose one or more Java objects since a component can be more than just an object. Enterprise JavaBeans has proved to be the best solution which is applied for component development environment [10].

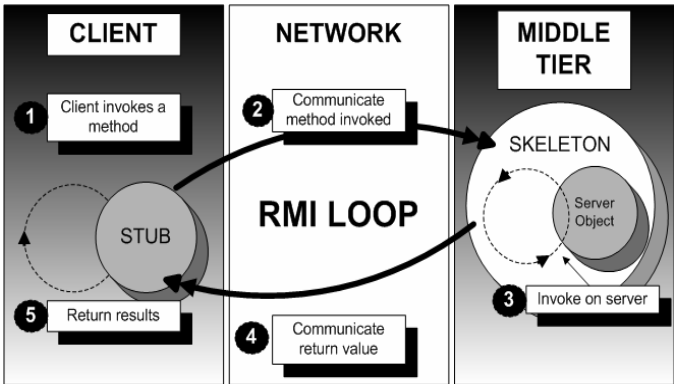

Fig.1 Enterprise JavaBeans [8]

No matter how an EJB is built, the same component interface is always used [9]. Both the EJB and the interface must follow the EJB-specification, which requires EJB's to expose certain methods that must be included. These methods allow the container to handle EJB's uniformly, no matter what container the EJB's are hosted in.

1) *THE EJB TECHNOLOGY:* There are three types of EJB's [9] that can be used, all with different features. The three bean types are:

- **Session beans (SB)**. This bean models a business process. It can be compared to a verb, because like a verb it is doing something. The things that it does can consist of

calculations, connections to data bases or to other EJB's etc. A SB can be either stateless or stateful.

- **Entity beans (EB)**. Opposite to a session bean, an entity bean models business data. EB's are similar to nouns in that they are data objects, like a product, an order, an employee, a credit card, a storage, etc. To be able to perform its task, all EB's are persistent. A SB usually harnesses an EB to achieve business goals. An example can for instance be a stock trading system where a stock trading engine is dealing with stocks. In this example the stock trading engine is the SB and the stocks are represented by one or more EB's.
- **Message-driven beans (MB)**. Message driven beans are similar to session beans in that they do something. The difference is that a message driven bean can only be called by using messages.

The reason for having three types of beans is because several businesses have been involved in the development of the EJB standard. The three bean types are a result of different requirements stated by those businesses. The requirements are based on the technologies they use and on their respective types of distributed systems. Having SB's, EB's and MB's provide maximum flexibility to support the different requirements.

EJB allows development of reusable components [11], this can be understood with the help of example where a credit card-charging module has been implemented as EJB component that is accessed by multiple applications as shown in Fig. 2.

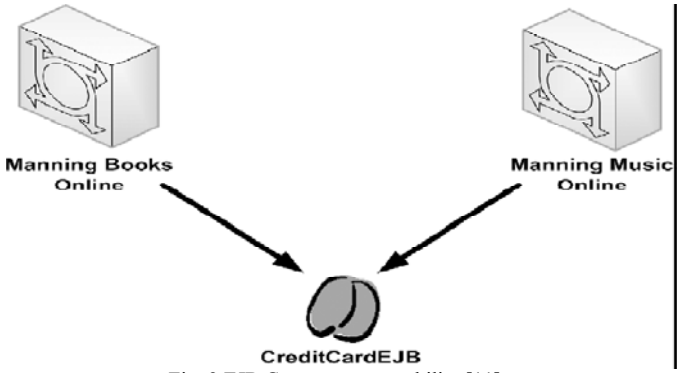

Fig. 2 EJB Component reusability [11]

The EJB architecture is a standard component server side architecture that is mainly used for server side development for building distributed object-oriented business applications in the Java programming language thus supporting the development, deployment, and use of web services in a layered architecture pattern[10]. Layered architecture implies the concept of components that are grouped into different levels, each level in the application serves a specific purpose. Each level in the application has a well-defined purpose, sort of like a section of a factory assembly line. Each section of the assembly line performs its designated task and passes the remaining work down the line.

EJB supports two types of layered architectures for developing applications – *the domain-driven design* (DDD) and *the traditional four-tier architecture.* The traditional four-tier server architecture is presented in this paper taken from [11] as shown in Fig. 3. In this architecture services for graphical user interface (GUI) and handling user input are offered by the presentation layer, thus passing down each request for application functionality to the next layer i.e. business logic layer. The business logic layer contains work flow and processing logic; it retrieves data from and saves data into the database by making use of persistence level. The business logic layer can perform some actions like billing, ordering, and maintenance of user account, etc. The persistence layer is responsible for providing high-level object-oriented (OO) abstraction over the database layer. The database layer consists of a relational database management system (RDBMS) like DB2, Oracle, MySQL, SQL Server, INGRES.

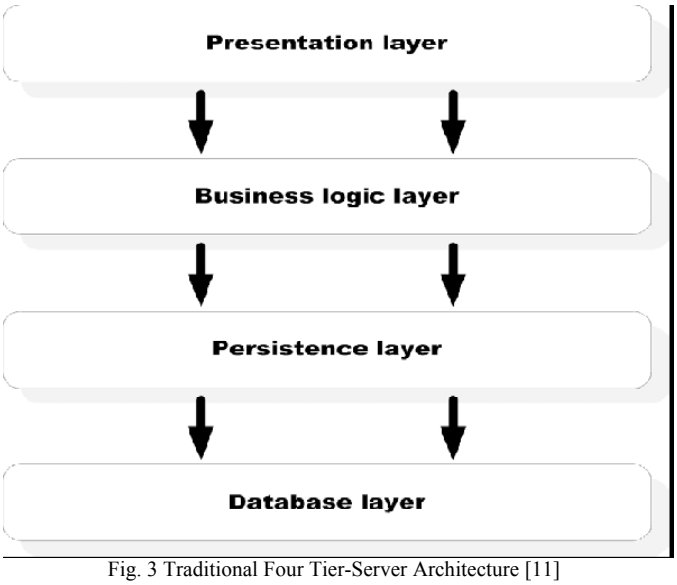

Fig. 3 Traditional Four Tier-Server Architecture [11]

Using EJB architecture it is possible to build applications by combining components developed using tools from different vendors and provide interoperability between enterprise beans and J2EE components as well as non-Java programming language applications [12]. EJB provides strong support for implementing the business logic and persistence layers. EJB servers give a bunch of services, so that we don't have to write them ourselves [13]. Fig. 4 represents some supporting services with in different layers like security, transaction management, concurrency, networking, resource management, persistence and messaging etc.

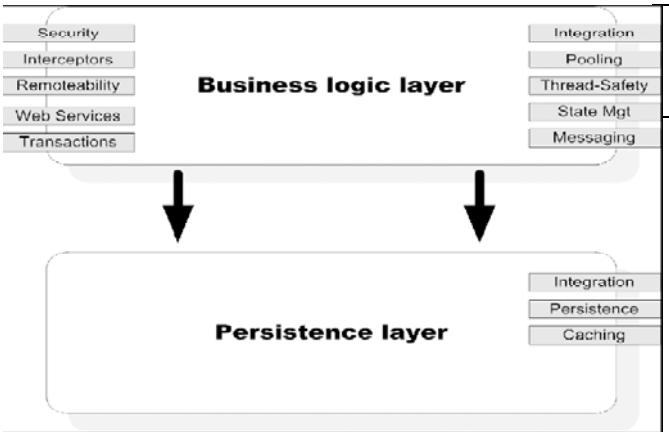

Fig. 4 Services provided by EJB in different layers

2) *ADVANTAGES AND LIMITATIONS OF EJB* [14]

Pros	Cons
• EJB is a standard and probably the simplest server-side development platform around. The features that shine the brightest are POJO programming, heavy use of sensible defaults, and Java persistence API etc [11].	• Learning EJB systematically is difficult. Using EJB in a wrong way may lead to poor presentation.
• EJB is based on Java programming language. Thus inheriting all advantages of Java, like Java does not have concept of pointers, garbage collector is present, memory leaks are virtually none.	• EJB is based on Java programming language. All disadvantages that Java has also exist in EJB.
• The EJB container handles beans uniformly as the beans are hosted in container thus providing communication handling, security, pooling. The developer time is saved in developing these features.	• To fully utilized containers computer system with higher end is required, or it may result in low performance.
• Component reliability and stability increases as threads are not used in beans, due to which programming are less complex.	• Threads have certain advantages and helps in solving certain tasks, but threads are not used in a bean.
• EJB technology has an added advantage of having Message driven beans.	• Security cannot be applied to Message driven beans.
• EJB having equipped with three beans is very flexible.	• In EJB every session bean consists of at least three Java classes, while every entity bean comprises at least four. Some standard and possibly vendor-specific deployment descriptors are needed. Here coding time can be reduced by some auto generation tools, thus restricting to a particular vendor [15].

B. CORBA

The *Common Object Request Broker Architecture* (CORBA) was proposed by the Object Management Group (OMG) as a standard for communication between components. CORBA is a standard for writing object oriented distributed systems [9]. This standard is platform independent. It forms part of a larger model, Object Management Architecture (OMA) that defines at a high level of abstraction a context within which components operates and interacts, including standard services that components can rely on [16].

1) CORBA TECHNOLOGY: CORBA is widely used in component-based software systems because it offers a consistent distributed programming and run-time environment over common programming languages, operating systems, and distributed networks [17]. That means software written in java, C++ etc. that may be running on different operating systems, can be transparently integrated with each other. The *Object Request Broker (ORB) is the* most important part of a CORBA system [17]. Client server relationship between components is established by ORB which acts as middleware. Fig. 5 illustrates the architecture of remote invocation with ORB [18]. With the help of ORB a method can be invoked by a client, whose location is completely transparent. The ORB performs the task of intercepting a call and finding an object that can implement the request, pass its parameters, invoke its method and return the results. The client does not need to know where the object is located, its programming language, its operating system, or any other system aspects that are not related to the interface. Thus interoperability is provided by ORB to synchronize applications on different machines in various distributed environments and truly interconnects multiple object systems.

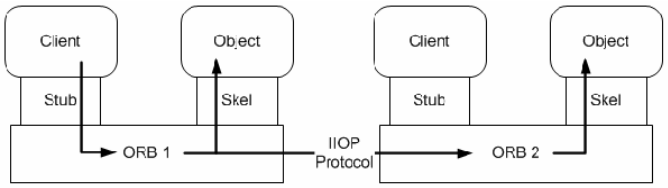

Fig.5 Remote Invocation in CORBA [18]

2) ADVANTAGES AND LIMITATIONS OF CORBA [14]

Pros	Cons
• It is a standard for writing object oriented distributed systems [9]. This standard is platform independent.	• There is a long time span between updates, taking years for a new version release. If there is a bug in CORBA, risk is that it will stick around next release.
• A standardized way of connecting and using	• Learning CORBA is difficult, especially when there are

components through interfaces.	other technologies that can do what is requested and are easier and faster to learn.
• It consists of numerous features for developing products.	• Products that are developed in CORBA may have incompatible features.
• CORBA appears to be reliable.	• CORBA does not support all of the reliability evaluation criteria [19].
• CORBA's flexibility gives the developer a countless number of choices.	• These choices require a vast number of details to be specified which increases the complexity too high to be able to do so efficiently and quickly [20].

C. .NET

The name .NET is used by Microsoft to denote a comprehensive set of new technologies. This includes a new component model, intended to replace COM/DCOM. A notable development is that .NET moves the responsibility of providing certain functionality from the components to a more sophisticated run-time system. In particular, COM/DCOM requires components to provide a considerable amount of "house-keeping" functionality that is taken care of by the .NET run-time. Much of the flexibility that follows from having such implementations in each component is maintained under .NET, where components can affect the operation of the run-time by setting declarative attributes [21].

.NET has been tailored in such a way that practically any programming language should be able to be compiled for .NET. This can be possible because a common base has been defined in .NET, which is the *Intermediate Language (IL)* [22]. When a developer compiles a .NET application or component etc. it is compiled to IL-code, which is interpreted by the *CLR (Common Language Runtime)* when running the application, or using the component, the *CLR* is the control centre of .NET. The .NET applications are virtually programming language independent, and to some extent platform independent as well. As long as the programming language is compiled to IL-code, it will work with the .NET environment. The .NET environment can be ported to other platforms other than Windows, to achieve platform independence like in Java.

1) .NET TECHNOLOGY: The heart of .NET is the CLR [23], as it contains a number of *JIT-compilers (Just-In-Time compilers)* that compile IL-code to native code. Fig. 6 shows how the CLR controls the .NET architecture.

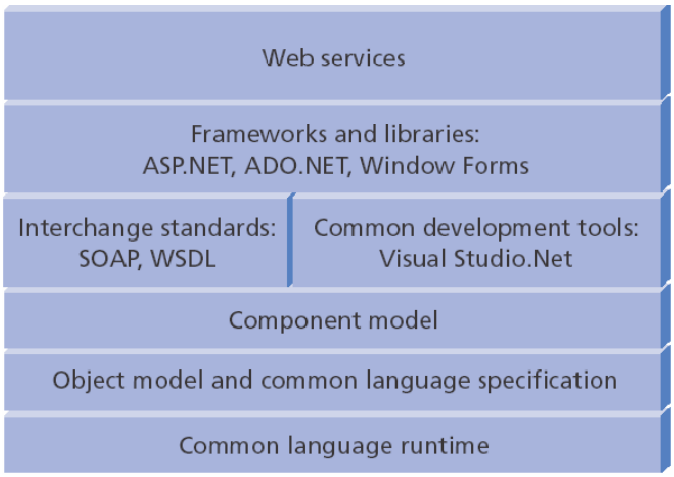

Fig. 6 The .NET Architecture [23]

When a .NET program is generally compiled it is compiled to a processor-independent one or more files consisting of IL-code [22]. The application is not compiled to native code until it is actually run. One of the JIT compilers which are part of the CLR handles the compilation automatically. All code that requires the CLR to be able to run is called *managed code*. The execution model of .NET can be well understood with the help of Fig. 7 which illustrates the running and compilation of a .NET application [22].

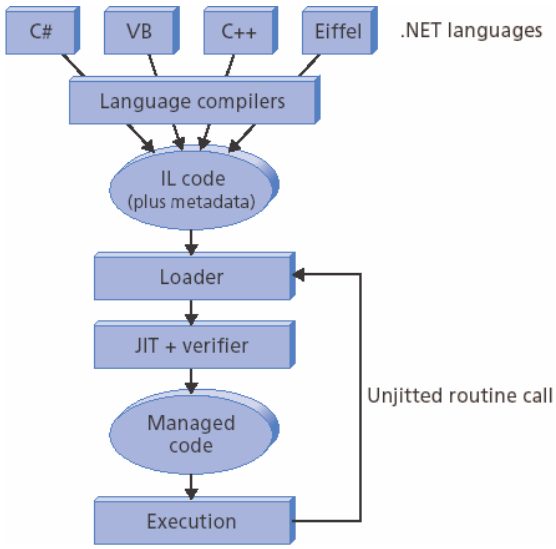

Fig.7 The .NET program execution model [22]

To save time during start-up of an application only the needed portions of the application is compiled by the CLR. When another part is needed, it is compiled at that time. Once some part of the application is compiled, its executable is cached until the application is closed. That way it does not have to be recompiled when it is in use [23]. The caching and the fact that only needed portions of the application are compiled make running and using a .NET application almost as fast as running a traditional executable.

Due to JIT-compilation the .NET applications are not that much faster compared to traditional applications even if it is close, there are large benefits by having the CLR compile the application each time it is run. The executable created by the CLR can be optimized for the platform hosting the CLR, and the application. The CLR checks all IL-code before it is compiled this increases the level of security as illegal actions can be prevented before they even get a chance to be executed. Besides from security and cross-language integration, CLR also handles *versioning* and deployment support, debugging and profiling services and memory management. A feature of .NET is that memory is automatically freed by a so called *garbage collector* when not in use anymore [23].

2) ASSEMBLIES: Assemblies are used for the building of .NET applications [23]. For a .NET application to work, all its classes must reside within one or more assemblies. All .NET components are stored in assemblies as well. They are however not components themselves, but can contain one or more of them [22, 23]. Assemblies can be stored in different ways other than in just an ordinary file, and should therefore rather be thought of as logical units than physical ones. The assembly is normally represented by files with the extension .exe or .dll. Both these file-types are structured in the same way, only the .exe-file has a startup sequence so the CLR knows how to start its execution [22]. An assembly provides the CLR with the information it needs to be aware of type implementations.

3) METADATA: Each component that is stored in an assembly, and also the assembly itself, has its own metadata [22, 23]. The metadata has a specific post for storing the version of the component. In the metadata, information is saved about each file that belongs to the assembly. If the assembly is divided into more than one file and one of those files where altered or replaced, the assembly would probably fail to load, since the saved information did not match the file information in the altered or new file. The *manifest file* of an assembly contains the metadata of that assembly, and it can be stored in an .exe-file or in a .dll-file [23].

3) ADVANTAGS AND LIMITATIONS OF .NET [14]

Pros	Cons
• .NET can be integrated in Windows operating system, giving high performance.	• .NET having its dependency mostly over Windows operating system may not work as well in other operating systems.
• .NET is safe, secure, faster and reliable for enterprise applications.	• From operational point of view the capacity and ongoing cost to support .NET is higher.
• Run-time system is reliable.	• .NET being a sophisticated run-time system, possibly without using much of its functionality, may lead to unnecessarily large software.
• Support for any programming language modification is done through IL-code so that applications developed in that language can work as .NET applications.	• Adapting to IL-code features, programming languages may lose features that made the language unique.
• CLR being a part of .NET, all applications work through it, that way security is improved.	• .NET applications take longer time to startup, because each time an application must be compiled before it is run.
• .NET supports multiple languages such as Visual Basic.NET, C# (pronounced as C-sharp), Jscript.NET and Managed C++ (a dialect of Visual C++). The beauty of multi language support lies in the fact that even though the syntax of each language is different, the basic capabilities of each language remain at par with one another [13].	• Future directions are determined by Microsoft means any application built on .NET using supporting languages are committed and dependent on Microsoft as license costs continue to increase with the enhanced restrictions in terms of use [13].

Table-1 describes the functionality and mechanism of each component technology. The Name Service provides a convenient way to find the component implementation associated with a particular public name. But its greater significance is that it allows an application to use multiple different implementations of a component at one time [23].

Remote communication in .NET uses its own self-hosted server, the result may vary depending on what container is used. Different vendors of EJB containers most likely have different ideas of how things should be done, like what algorithms are used for handling instance pooling, or what policies are best regarding security or lifetime management. In some cases, a container may not even support certain features of the EJB technology [14].

The execution semantics of an invoked component operation describes the context in which that operation executes: it could execute in the caller's thread or the callee's (component's) context, where there could be one thread reserved for each caller, or new thread could be used for each operation invocation [23].

A component interface is important for determining the scope of a component, all models have interfaces with EJB having explicit java interface, CORBA has language independent Interface Definition Language and .NET has explicit language interface.

Granularity of component, for granularity the question is whether a component is a single class, a single package, or a set of packages. In EJB and .NET the granularity of component is EJB uses EJB-jar files, and .NET uses the Assembly DLL. For CORBA the typical scope and granularity of a component is less clear. It is often treated like a COM component, but given the complete lack of guidance, any of the choices is possible [23].

TABLE-1: COMPARISON OF COMPONENT TECHNOLOGIES [14] [23]

	EJB	**CORBA**	**.NET**
Functionality			
Name Service	public name	public name	public name
Remote communication	Transparent	transparent	Transparent
Execution semantics of invoked component operations.	dictated by bean type (entity, session, message beans)	one thread per invocation; executes in component context	various choices
Mechanism			
Interface specification	explicit java interface	language-independent IDL	explicit language interface
Granularity of component	EJB-jar file	various	Assembly DLL file
Component packaging standards	EJB-jar file	none	Assembly DLL file
Technology integration	EJB technology integrates well with CORBA.	CORBA and EJB can be integrated with each other.	.NET applications can use CORBA, but it is not as well integrated as in the EJB technology.

Table-1 summarizes key features of Component models (EJB, CORBA, .NET) [14] [23

The lack of a standard for packaging (preparing and delivering) creates ambiguity about the granularity of a component: in some cases multiple granularities are possible, and in other cases it simply causes confusion. In EJB and .NET there is a packaging standard that specifies a deployable unit: EJB uses EJB-jar files, and .NET uses the Assembly DLL [23].

The EJB technology is in fact based on CORBA concepts. EJB and CORBA can be integrated with each other [14].

IV. CONCLUSIONS

Component Based Software Development is the latest advance [6] in software development, promising the possibility of extending the real world approach to create well-specified parts and top incorporate legacy code "wrapped" as components. Affordability, simplicity, reliability, adaptability, reduced workload are the major scores for its success.

An attempt has been made to give a clear overview of component-based software development, by discussing some major activities involved in CBSE. Three main technologies used for CBSD along with their advantages and limitations are discussed in this paper like EJB, CORBA, .NET covering in depth as they are more popular in software industry. A comparison of these technologies based on functionality and mechanism is conducted which will definitely provide the reader an in depth understanding and guidance in selecting these technologies for CBSD.

V. REFERENCES

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

[2] J.W. Shareef, Component-Based Software Development: An Appropriate and Methodical Approach, International Journal for Electro Computational World Knowledge Interface, Vol.1, Issue 5, Jan. 2012, ISSN No. 2249-541X.

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

[4] Brown, W. Alan & C. Kurt Wallnau, "Engihttp://cbs.colognet.org/ overview.php Engineering of Component-Based Systems," 7-15. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.

[5] Rogers, S. Pressman, Ph. D., "Software engineering- Practitioner´s approach" 5th Edition, McGraw-Hill series in computer science, 2001.

[6] G.T. Heineman, and W.T. Councill, Component-Based Software Engineering: Putting the Pieces Together, Addison-Wesley, May 2001.

[7] M. Casanova, R.V.D. Straeten, and V. Jonckers, Supporting Evolution in Component-Based Development using Component Libraries. http://distrinet.cs.kuleuven.be/projects/ SEESCOA/publications/Casan--ova.pdf, last access: 26-12-11.

[8] Monson-Haefel Richard, *Enterprise JavaBeans*, O'Reilly, 2001.

[9] E. Roman, S. Ambler, T. Jewell, Mastering Enterprise Java Beans Second Edition, Wiley Computer Publishing, 2002.

[10] L. DeMichiel, Sun Microsystems, Enterprise JavaBeans™ Specification, Version 2.1, Sun Microsystems, pp.1-635, 2002.

[11] P. Debu, R. Reza and L. Derek, "EJB 3 in action", Manning publication, 2007.

[12] M.H. Selamat, H. Sanatnama, A.A.A. Ghani and R. Atan, Software Component Models from a Technical perspective. IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.10, pp.-135-147, October 2007.
[13] S.A. Khan, W. Hussain, Component Based Software Development with EJB and .NET, Mälardalen University, Department of computer science and electronics Västerås –Sweden, 2008.
[14] J. Persson, Comparison of Enterprise Java Beans and .NET from a Component Point of View, Master Thesis, Software Engineering, Thesis no.MSE-2003:29, pp.1-98, 2003.
[15] H. Sheil, "To EJB, or not to EJB?" http://www.javaworld.com/javaworld/jw-12-2001/jw-1207-yesnoejb.html?last access: 27-12-2011.
[16] P. Cox, B. Song, "A Formal Model for Component-Based Software", IEEE 2001 Symposia on Human Centric Computing Languages and Environments, Stresa, Italy, September 05 - 09, 2001.
[17] C. Xia, LYU R. Michael, W. Kam-Fai, K.O. Roy, *Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes*, http://www.cse.cuhk.edu.hk/~lyu/paper_pdf/apsec.pdf, last access: 15.09.2011.
[18] ÖZYURT Barış, *Enforcing Connection-Related Constraints And Enhancements On A Component Oriented Software Engineering CASE Tool*, Master Thesis. 2003.
[19] J. Guo, Y. Liao, Assessment of Component-Based Systems with Distributed Object Technologies, *http:// ww1.ucmss.com/books/LFS/CSREA2006/SER3741.pdf, last access : 25-12-2011.*
[20] OMG, "CORBA Components," Report ORBOS/99-02-01, OBJECT MANAGEMENT GROUP, 1998, http://www.omg.org.
[21] F. Lüders, Use of Component-Based Software Architectures in Industrial Control Systems, Thesis, pp.-117, 2003.
[22] S. Robinson et.al, Professional C# 2nd Edition, Wrox, 2002.
[23] C. James, Introducing dotNET, Wrox, 2000.
[23] Jr. W. DePrince, C. Hofmeister, System Design, Development, and Maintenance, Proceedings of the 3rd Working International Conference on Software Architecture, August 2002, pp. 205-219, 2002.