IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Vol.8, Issue.9, September 2020 E-ISSN: 2347-2693

Component Based Software Development using Distributed Objects

Sanjay E. Yedey

Department of Computer Science and Technology, DCPE, Amravati, INDIA

Author's Mail id: sanjayeyedey@gmail.com

DOI: https://doi.org/10.26438/ijcse/v8i9.7984 | Available online at: www.ijcseonline.org

Received: 20/Aug/2020, Accepted: 05/Sept/2020, Published: 30/Sept/2020

***Abstract*-** The Object Oriented Programming paradigm has revolutionized the process of software development. It provides a great control over data and offers various revolutionary features like abstraction, encapsulation, polymorphism, inheritance that facilitates reusability of previous efforts done to build softwares. This approach makes it possible to develop softwares as reusable component that can be assembled with other. This software development paradigm makes it possible to develop software applications based on „plug and play" in which we can add, replace or modify components according to our needs. This Component Based Software development approach provides a cost effective, fast and modular approach for developing complex software with reduced delivery time. The technologies that facilitate Component development are distributed object technologies. The Distributed Object technology allows objects active in one process be accessed by another facilitating the computation be split over multiple processes. The processes involved may be running in different address spaces on single system or may be on different systems in a network in a local area network or the Internet. The most popular distributed object technologies are CORBA, RMI and DCOM.
This paper presents an analysis of architecture and working of these technologies and compares software development methodology in these technologies on the basis of key terminologies used such as data marshalling, interoperability, heterogeneity, design transparency and speed.

***Keywords*-** Software Components, Components based Software Development, Distributed Objects, CORBA, RMI, Marshalling, Interoperability, Heterogeneity, Design Transparency

I. INTRODUCTION

Software development Software development process has evolved a long way from traditional waterfall model to highly manageable component oriented software. Earlier softwares development was done using procedural approach in which softwares were built by breaking functional requirements into sub tasks and building a software for each individual task. Later on the software development process witnessed a big leap with the introduction of object oriented paradigms. The Object Oriented Programming System (OOPS) provided a revolutionary approach where the main focus was on data and entities rather than on functions. It provides a great control over data and offers various revolutionary features like abstraction, encapsulation, polymorphism, inheritance that facilitates reusability of previous efforts done to build softwares. This approach makes it possible to develop softwares as reusable components. Each component represents a set of services which can be assembled with other [1,2]. This software development paradigm makes it possible to introduce plug and play approach in building software applications in which we can add, replace or modify components according to our needs. This helps in reducing software crisis and delivers robust software products with faster delivery and reduced cost. The advantage of this Component Based Software Development approach is that it provides a cost effective, fast and modular approach for developing complex software with reduced delivery time. The code reuse in designs allows taking advantage of the investment done on earlier reusable components. Another advantage is that, a software component can be developed and deployed independently and is subject to be composed by third party[3]. Components are built to be reusable which makes development of further applications with similar functionalities much easier. Components are heterogeneous in nature in terms of programming languages and platforms[4].

Characteristics of Component Based Systems

- Component-Based Architecture (CBA) is based on the principle "Collaboration via Cooperation", in which a set of individual components, having their own goal, contribute their efforts to collective bigger goal. Every component is capable of functioning independently. A big problem is decomposed into smaller sub-problems and individual reusable software components are designed to solve each of such smaller problems. The Component based software design provides a higher level of abstraction than that of Object Oriented Design principles. Some of the major characteristics of CBA are given below. The Figure-1 depicts these design principles.

1. Reusability: The first and foremost characteristic of CBA is that the components are designed to be reusable in different applications, so that "Write once, Use Anywhere" principle is achieved.

2. Extensibility: Components are designed in such a way that, their capabilities can be extended without being needed to disturb existing designs.
3.Encapsulation: Components advertise their capabilities via interfaces which allow application developers to use their functionality without revealing its proprietary or vital details.
4. Independent: Each components is an individual and independent unit, though it can be used in a society of other components to achieve collective bigger goal **.**
5. Replaceable: If needed Components can be substituted easily with other similar components with enhanced capabilities.

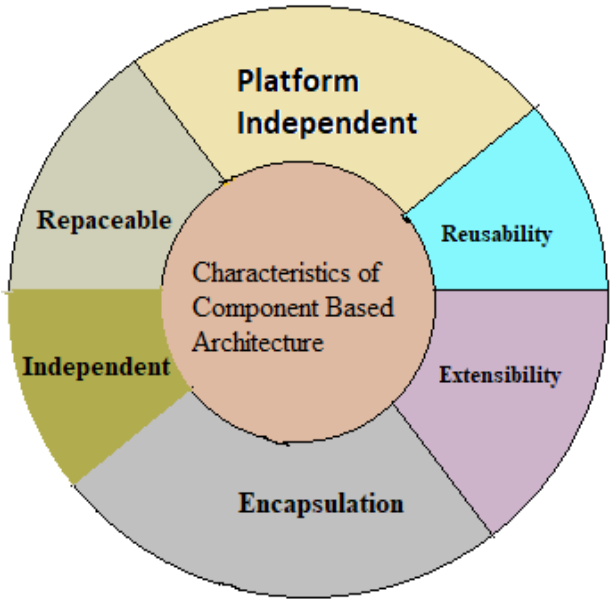

Figure-1: Characteristics of Component Based Architect

6. Platform Independent: Components are designed to function in heterogeneous environments, no matter in which language they are developed and which platform they get to operate in.

With the growth of the Internet and the advancement in communication technology many applications useful in our day to day life are made available on the internet and are accessed at ease using portable devices like smart phones and laptops along with micro to mainframe or even supercomputers. This creates an environment of heterogeneous distributed systems spread across all over the world. The requests from large number of users create a tremendous burden on the server, affecting performance of the services on web server. Although a centralized system approach facilitated by powerful systems like supercomputer or mainframe computer was employed to solve the problem at an early stage, it had its own limitations with respect to aspects like bandwidth, traffic congestion, reliability and even cost. The limitations made client/server and distributed processing approach more suitable[5,6]. Client/Server network uses a network operation system designed to manage the entire network from a centralized point, which is the server. Clients make requests of the server and the server responds with the information or access to a resource[7]. In addition, the fast growth of a network performance has accelerated the multiple computer approaches. A distributed processing is widely used these days, especially in multi-tier environments.

Among the distributed processing technologies, Remote Procedure Call (RPC) [8,9] and Remote Method Invocation (RMI) [10] exist as early models. However, as the object oriented paradigms flourished, these models have beenevolved into distributed object technologies. The established off-the-self Distributed Object Technologies include Microsoftt‟s s Distributed Component Object Model (DCOM), Object Management Groupg‟s Common Object Request Broker Architecture (CORBA) and that of Java‟s Remote Methos Invocation(RMI)[11,12,13].

This paper describes the architecture and working of these technologies along with advantages and disadvantages of each technology, providing a guideline to the developers, vendors, and practitioners to help them to choose an appropriate technology to develop mission critical application in distributed environment.

II. RELATED WORK

The principle idea behind Component Based Software development approach is building applications from pre-existing and precompiled software components that are available in executable binaries or byte codes. Ivica C, Stig L and Michel C [14], observes that, this idea has a big impact on the system development lifecycle. First, the developer has to separate system development process from that of the components; the components should already have been developed and possibly used in other products when the system development process starts. Second, the activities in the processes will be different from the activities in non-component based approach; for the system development the emphasis will be on finding the proper components and verifying them, on the other hand, for the component development, design for reuse will be the main concern. Wallis, Henskens, Hannaford, and Paul[15], found from a software development perspective, it is observed that it is possible to re-use existing Web components within the new distributed component-based framework by wrapping them within component interfaces. It is also possible that, once the runtime environment and system-level components are available, the developer needs only be concerned with defining the interfaces and building the application-specific components required for their application. The runtime environment handles automatically the complexity of the distributed nature in which the components are executing.

Ivica and Magnus, [6] observes, since most applications need to be modified from time and again as per changing needs, the components utilized to build the application must be maintained or replaced. The evolution of

requirements affects not only specific system functions and particular components but also the overall component-based architecture at every level. Increased complexity is a consequence of different components and systems having different life cycles. In component-based systems it is easier to replace part of system with a commercial component.

III. DISTRIBUTED SOFTWARE SYSTEMS

A distributed system consists of a collection of autonomous computers linked by a computer network equipped with distributed system software. This software enables computers to coordinate their activities and to share the resources of the system hardware, software and data. Users of a distributed system should perceive a single, integrated computing facility even though it may be implemented by many computers in different locations. This is in contrast to a network, where the user is aware that there are several machines whose locations, storage replications, load balancing and functionality are not transparent[17,18]. Benefits of distributed systems include bridging geographic distances, improving performance and availability, maintaining autonomy, reducing cost and allowing for interaction.

IV. DISTRIBUTED OBJECTS AS SOFTWARE COMPONENT SYSTEMS

In distributed system technologies, the concept of D**istributed Objects'** refers to a technique in which objects are distributed across different address spaces, either in different processes on the same computer, or even in multiple computers connected via a network. Distributed object models and tools extend an object-oriented programming system. These objects, though active on different machines, work together in collaboration by sharing data and invoking methods[18]. This communication often involves location transparency, where remotely located objects appear the same as local objects. The principal way of communication among these distributed objects is by using „Remote Method Invocation (RMI)", generally by message-passing. In message-passing, one object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

The objects may be distributed on different computers throughout a network, living within their own dynamic library outside of an application, and yet appear as though they were local within the application. This is the essence of plug-and-play software. Several technical advantages result from a distributed object environment. The overall technical goal of distributed object computing is to advance distributed information technologies so that they may be more efficient and flexible, yet less complex[19]. The benefits of distributed objects are indeed solutions to the problems with existing, monolithic client/server paradigms.

A. CORBA

CORBA is part of the *Object Management Architecture* (OMA), developed by OMG, which is also the broadest distributed object middleware available in terms of scope. It allows integration of a wide variety of object systems. The basic OMA reference model from the OMG specification presents CORBA architecture. The *Object Request Broker* (ORB) component enables clients and objects to communicate in a distributed environment[18]. Four categories of object interfaces use ORB to interact:

- *Object Services* are interfaces for general services that are likely to be used in any program based on distributed objects.
- *Common Facilities* are interfaces for horizontal end-user-oriented facilities applicable to most application domains.
- *Domain Interfaces* are application domain-specific interfaces, which may also be a collection of different Domain Interfaces such as Finance, Telecom, Transportation, etc.
- *Application Interfaces* are non-standardized application-specific interfaces.

The key component in OMA is ORB, or specified as CORBA. From the above description, it is not hard to see that ORB needs to provide the functions of delivering requests to objects and returning any responses to the clients targeted. As a distributed environment, ORB shall also support the transparency requirement. CORBA presents a nice architecture of an ORB, which handles a series of jobs like object allocation, object implementation, object execution state, object communication mechanisms, etc. Following the CORBA architecture, most of the jobs to delivery object communication are transparent. In this sense, the four categories of OMA objects can be connected to a CORBA ORB to form a distributed computing environment without worrying about any of the communication issues among them. Above Figure demonstrates CORBA ORB architecture with its key components. Here, we try to understand CORBA structure through studying some of its key components. Some features that are important to CORBA are also discussed below.

B. DCOM V. DCOM is more or less an architecture specification designed to be language-independent. DCOM is primarily implemented on Windows platforms, and specified by Microsoft [11,12]. Microsoft DCOM is often called ``COM on the wire''. It supports remote objects by running on a protocol called *Object Remote Procedure Call* (ORPC).

A DCOM client calls into the exposed methods of a DCOM server by acquiring a pointer to one of the server object's interfaces. The client object then starts calling the server object's exposed methods through the acquired interface pointer as if the server object resided in the client's address space. Since the COM specification is at the binary level it allows DCOM server components to be

written in diverse programming languages like C++, Java, Object Pascal (Delphi), Visual Basic and even COBOL. As long as a platform supports COM services, DCOM can be implemented on the platform. However, it is practically not available except Windows systems.

C. RMI

An object-based programming language encourages a methodology for designing and creating a program as a set of autonomous components, whereas a distributed operating system permits a collection of workstations or personal computers to be treated as a single entity. Java RMI provides an elegant and powerful model for invoking member functions on objects that exist in remote address spaces. Sun Java RMI is a built-in native ORB in Java language. It supports making method invocations on remote objects. From practical programming point of view, developing distributed applications in RMI is simpler than developing with sockets since there is no need to design a protocol, which is an error-prone task. In RMI, the developer has the illusion of calling a local method from a local class file, when in fact the arguments are shipped to the remote target and interpreted, and the results are sent back to the callers. The underlying protocol for RMI is *Java Remote Method Protocol* (JRMP).

Feature Analysis and Semantic Comparison: As distributed object technologies the architectures of CORBA, DCOM and Java/RMI provide mechanisms for transparent invocation and accessing of remote distributed objects. Though their objective and approach is more or less same the mechanisms that they employ to achieve remoting and many other issues with respect to their central objective is a lot different. The Table-1 provides detailed comparisons based on different aspects.

Comparison of Distributed Object Technologies based on Component Development Metrics:

1. **Base Interface (Base Type):**
   - **DCOM**: Every sever object implements IUnknown interface
   - **CORBA**: Every sever object implements CORBA.Object
   - **JAVA RMI**: Every server object implements java.rmi.Remote
2. **Interface Definition Language (IDL):**
   - **DCOM**: The Microsoft"s MIDL DCOM interfaces. The MIDL compiler creates proxy stubs for the client and server
   - **CORBA**: CORBA IDL defines the methods and attributes of component interface The IDL compiler creates proxy stubs for the client and server.t
   - **JAVA RMI**: RMI defines interfaces in Java language. RMIC compiler is used to compile and create proxy stub and skeleton
3. **Unique Identification**
   - **DCOM**: interface – an interface is uniquely identified by an id called IID named implementation of the server object is uniquely identified by id called CLSID
   - **CORBA**: interface – an interface is uniquely identified by an interface name, named implementation of server object is uniquely identified by its mapping to a name in the Implementation Repository
   - **JAVA RMI**: interface – an interface is uniquely identified by an interface name named implementation of the server object is uniquely identified by its mapping to a URL in the Registry
4. Remote Object Reference (object handle at run-time)
   - **DCOM**: Uniquely identifies a remote server object through its interface pointer, which serves as the object handle at run-time
   - **CORBA**: Uniquely identifies remote server objects through object references (objRef), which serves as the object handle at run-time
   - **JAVA RMI**: Uniquely identifies remote server objects with the ObjID, which serves as the object handle at run-time.
5. Object Handle
   - **DCOM: Interface Pointer**
   - **CORBA:** Object Reference
   - **RMI:** Object Reference
6. Remote Object Reference Creation
   - **DCOM:** Generated by Object Exporter
   - **CORBA:** Generated by Object Adapter
   - **RMI:** Generated by the call to the method UnicastRemoteObject exportObject (this)
7. Object and skeleton instantiation
   - **DCOM:** Tasks like Object and skeleton registration are performed by server program or handled dynamically by the COM run-time system.
   - **CORBA:** Tasks like Object and skeleton registration are performed by
   - **RMI:** Object registration is done through RMIRegistry using Naming class. skeleton registration is done by its instantiation on calling UnicastRemoteObject.exportObject(this) method
8. Underlying Remoting Protocol
   - **DCOM:** Object Remote Procedure Call(ORPC)
   - **CORBA:** Internet Inter-ORB Protocol(IIOP)
   - **RMI:** Java Remote Method Protocol (JRMP)
9. Object Activation
   - **DCOM:** Client calls CoCreateInstance()it needs a server object
   - **CORBA:** Client binds to a naming or a trader service when it needs server object
   - **RMI:** Client calls lookup() on the remote server object's URL name when it needs server object
10. Handle Mapping of Object name to Object Implementation
    - **DCOM:** Handled by the windows Registry
    - **CORBA:** handled by the Implementation Repository
    - **RMI:** Handled by the RMIRegistry
11. Type Information for methods
    - **DCOM:** Stored in Type Library
    - **CORBA:** Stored in Interface Repository

- **RMI:** Any type information is held by the Object itself

12. Locating an object implementation
- **DCOM:** Handled by Service Control Manager (SCM)
- **CORBA:** Handled using Object Request Broker (ORB)
- **RMI:** handled Java Virtual Machine (JVM)

13. Parameter passing
- **DCOM:** All parameters passed between the client and server objects are passed either by value or by reference
- **CORBA:** All interface types are passed by reference. All other objects are passed by value including highly complex data types
- **RMI:** All objects implementing, "remote interfaces" extending java.rmi.Remote are passed by remote reference. All other objects are passed by value

14. Parameter Marshalling
- **DCOM:** Parameter marshalling is accomplished in the stub code that is generated by the IDL compiler. The client stub and server skeleton are responsible for marshalling of parameters.
- **CORBA:** DCOM provides automatic marshalling for primitive types and object references. For user defined structures and structured arrays Custom marshalling is preferred.
- **RMI:** RMI provides automatic marshalling of predefined types. and object references.Serialization is used for marshalling objects

15. Platform Independence
- **DCOM:** Runs on any platform having a COM Service implementation available on it.
- **CORBA:** Runs on any platform having CORBA ORB implementation available on it.
- **RMI:** Runs on any platform having Java Virtual Machine implementation available on it

16. Language Independence
- **DCOM:** No
- **CORBA:** Yes
- **RMI:** No, Only JAVA.

17. Exception Handling
- **DCOM:** Runs on any platform having Java Virtual Machine implementation available on it
- **CORBA:** Through Exception Objects
- **RMI:** Through RemoteException

18. Support for data and code reuse
- **DCOM:** Supports code reuse just by modifying the registry entry, without needing to recompile code on the client or server
- **CORBA:** Supports code reuse by writing CORBA compatible new objects
- **RMI:** Supports code reuse through Object Inheritense

19. Support for Multiple Inheritacne
- **DCOM:** Yes, at Interfaces as well as Object implementation
- **CORBA:** Yes, at interface level
- **RMI:** Yes, at interface level.

20. Security
- **DCOM:** Provided by NT Security
- **CORBA:** Provided by CORBA Security
- **RMI:** RMI security is provided by java security API.

V. CONCLUSION

The Distributed Object Technologies offer the techniques to develop softwares as 'Reusable Components'. These state of the art platform independent components which interact with each other through implementation-neutral interfaces allow us to develop software by assembling them into a single workable application unit. The methods of an object active on one machine can be accessed by remotely located programs in a smooth, secure and transparent manner. The two communication parties called client and server programmes may be running on heterogeneous platform both in terms of hardware as well as software. This paper presented a analytical survey of most popular Distributed Object Technologies, CORBA, DCOM and RMI. The survey conducted with respect to aspects like platform independence, language independence, marshalling, reusability, remoting protocol, security etc. and a comparison is presented which helps choosing technology most suitable for one‟s application.

REFERENCES

[1] Szyperski C, "Component Software-Beyond Object-Oriented Programming", Addison-Wesley, 1998.

[2] Tassio V, Ivica C, Eduardo S A, Paulo A M, Yguarata C C and Silvio R L M, "Twenty-Eight Years of Component Based Software Engineering", The Journal of Systems and Software, 111, pp. 128–
148, 2016

[3] Deepti N, Yashwant S C, Priti D and Aditya H, "An Analytical Study of Component-Based Life Cycle Models: A Survey", In Proceedings of International Conference on Computational Intelligence and Communication Networks (CICN), 2015

[4] Crnkovic I and Magnus L, "Component-Based Software Engineering-New Paradigm of Software Development", Invited talk and report, MIPRO, pp- 523-524, 2001

[5] Guynes C and Windsor J, "Revisiting Client/Server Computing", Journal of Business & Economics Research (JBER). 9. 10.19030/jber.v9i1.935. (2011).

[6] George C, Jean D, Tim K,"*Distributed Systems: Concepts and Design",* 5th edition, Addison Wesley, (2011).

[7] Namita V J, Snehal C P, Malhari R R, "Client Server Network Management System for WLAN (Wi-Fi) with Remote Monitoring", IJSRNSC, Volume-1, Issue-1, Apr- 2013. ISSN: 2321-3256.

[8] A.M. Khandker ; P. Honeyman ; T.J. Teorey,"Performance of DCE RPC", Second International Workshop on Services in Distributed and Networked Environments, 1995.

[9] Wesam A, Azzam S, Oraib A, Shatha Al-Asir, Shorouq A, "Interactive RPC Binding Model",European Journal of Scientific Research 27:1450-216 · January 2009

[10] "RMI Unleashes the Highest Performing Multi-core Processor and Product Family in the Industry, Driving System and Performance Scalability". *Press release*. RMI. May 19, 2009.

[11] Fabian Breg, Shridhar Diwan, Juan Villacis, Jayashree B, Esra Akman, Dennis Gannon Java RMI Performance and Object Model Interoperability: Experiments with Java/HPC++ Distributed Components, December 2012

[12] J. D. Schoeffler, "A Model For Estimating Overhead in DCOM and CORBA Function Calls", NASA Report, 1998
[13] Roger S C, Samuel T C, "Distributed, object-based programming systems", *ACM Computing Surveys (CSUR)Vol. 23, No. 1, March-1991*
[14] Ivica C, Stig L and Michel C, "Component-based Development Process and Component Lifecycle", Journal of Computing and Information Technology-CIT 13, 2005, 4, 321-327.
[15] Wallis, Henskens, Hannaford, and Paul, "Implementation and Evaluation of a Component-Based framework for Internet Applications", published in the journal IT in Industry, vol. 5, no. 2, 2017 ISSN (Online): 2203-1731
[16] Anandi Mahajan and Pankaj Sharma, "Object Oriented Requirement management Tools for maintaining of status of requirements", International Journal of Scientific Research in Computer Science and Engineering Vol.6, Issue.6, pp.27-30, December (2018) E-ISSN: 2320-7639.
[17] Ivica Crnkovic and Magnus Larsson,"Challenges of component-based development", published in the Journal of Systems and Software,(2002) 201–212.
[18] Elfwing, R., Paulsson, U., and Lundberg L., *Performance of SOAP in Web Service Environment Compared to CORBA*, In Proceedings of the Ninth Asia-Pacific Software Engineering Conference, IEEE, 2002.
[19] Remzi H and Andrea C, "*Introduction to Distributed Systems"*, Arpaci-Dusseau Books, 2014.

AUTHOR'S PROFILE

Dr. S. E. Yede obtained Bachelor of Science in Computer Science from Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur, India in 1991 and Master of Science in Computer Science from Dr. Babasaheb Ambedkar Marathwada University, Aurangabad, India in the year 1993. He obtained his Ph.D. degree in Computer Science from RSTM Nagpur University Nagpur in the year 2015.

He is currently working as Associate Professor, for Master In Computer Application (MCA) at P. G. Department of Compuer Science and Technology, from the year 1993 DCPE, affiliated to Sant Gadgebaba Amravati University, Amravati. He is a Life member of ISTE since year 2005. He has published more than 20 research papers in reputed international journals and conference procedings. His main research work focuses on Object Oriented Technologies, particularly Distributed Object Systems and Intelligent Agents. Recently he is working towards Big Data Analytics, Data Mining, IoT and Computational Intelligence based education. He has a vast experience of more than 25 years of teaching at post graduate level and more than 10 years of Research Experience.

