Component Object Model: An Overview & Practical Implementation

Article · March 2017

CITATIONS
0

READS
400

5 authors, including:

Sanjeev Kumar
Defence Research and Development Organisation
15 PUBLICATIONS **152** CITATIONS
SEE PROFILE

Alok Mall
Defence Research and Development Organisation
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Ratnakar Awasthi
University of Allahabad
4 PUBLICATIONS **17** CITATIONS
SEE PROFILE

K. C. Tripathi
Defence Institute of Advanced Technology
11 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Component Object Model: An Overview & Practical Implementation

Sanjeev Kumar[1], Alok Mall[2], Ratnakar Awasthi[3], KC Tripathi[4], Shubha Lakshmi[5]
[1]Scientist 'D', DMSRDE, Kanpur,
[2]Scientist 'G', DOP DRDO HQr, New Delhi,
[3]Scientist 'F', JCB, DRDO, New Delhi,
[4]Technical Officer 'A', DMSRDE, Kanpur,
[5]Avinashilingam University for Women, Coimbatore, Tamil Nadu (India)

Abstract- The Component Object Model (COM) is a system technology that originated with Windows, but has begun to propagate to other platforms (the Macintosh, Compaq/Digital VMS, Compaq Digital Unix, Solaris, other Unix flavors, mainframes, etc.) as well. Its purposes for software developers are multi-fold. The paper presents an overview of COM technology and concentrates on its advantages, which have made it one of the most versatile and indispensable technologies of today. The paper also carries a short description of an example project created using Microsoft Visual C++ based on COM and OLE features. The project automates MICROSOFT EXCEL, creating a worksheet and chart based on the input provided by the user. This model is being used for data analysis, visualization and data storage of wide range of materials being used for structural, stealth and polymer in defence application.

Keywords- Component Object Model, COM Technology, OLE, Active X, Distributed COM

I. INTRODUCTION

A. OLE – An Overview

Object Linking and Embedding (OLE) is a distributed object system and protocol developed by Microsoft. OLE is the technology which allows an object (e.g. a spreadsheet) to be embedded (and linked) inside another document (e.g. a word processor document).

The main benefit of OLE is that references to data in the master file can be made and the master file can then have the changed data which will in turn affect the referenced document.

Its primary use is for managing compound documents, but it is also used for transferring data between different applications using drag and drop and clipboard operations. The concept of "embedding" is also used for embedding multimedia in Web pages, which tend to embed video, animation (include Flash animation), and music files within the HTML code.

B. Evolution of OLE

OLE 1.0, released in 1991, was the first wildly adapted specification for developing component-document applications. It was the evolution of the original dynamic data exchange (DDE) concepts which Microsoft developed for earlier versions of Windows. While DDE was limited to transferring limited amounts of data between two running applications, OLE was capable of maintaining active links between two documents or even embedding one type of document within another.

OLE 1.0 later evolved to become architecture for software components known as the component object model (COM) which further graduated to Distributed Component Object Model (DCOM).

OLE 2.0

Released in early 1993, it provided a much richer compound document model (i.e. the containing multiple data types like text, video, graphics etc.), as well as OLE automation, OLE drag and drop and generic services.

At the core of OLE 2.0 is the Component Object Model (COM), a specification that allows developers to design interfaces that enable interaction among components. In fact, OLE 2.0 is simply a set of COM interfaces designed by Microsoft.

ActiveX

In 1996, Microsoft renamed the OLE 2.0 technology as ActiveX. This version of OLE is commonly used by Web designers to embed multimedia files in Web pages.

C. An Overview Of Com And Com Related Terminologies

A component software architecture from Microsoft, which defines a structure for building program routines (objects) that can be called up and executed in a Windows environment.

Some parts of Windows and Microsoft developed applications are also built as COM objects. COM provides the interfaces between objects, and Distributed COM (DCOM) enables them to run remotely.

COM was designed with C++ programming environment in mind. It supports encapsulation, polymorphism, and reusability. However, COM was also designed to be operating at compatible at the binary level and therefore is different from a C++ object. Generally, the high level programming languages such as C, C++, PASCAL, and ADA are machine-dependent. As a binary object, a COM object concentrates on its interface with other objects. When not used in the environment of its creator, an interface is exposed that can be seen in the non-native foreign environment. It can be seen because it is a binary object and therefore *not machine-dependent*. This does not require the host environment or an interacting object to know anything about the COM object. It is important to note that COM is not a programming language; it is a binary standard that enables software components to interact with each other as objects. COM is not specific to any particular programming language. COM can work with any language that can support the binary layout of a COM object. It is a programming model to facilitate the programmability related to this standard.

COM is used in the following ways:

1. COM Objects

COM objects can be small or large. They can be written in several programming languages, and they can perform any kind of processing. A program can call the object whenever it needs its services. Objects can be run remotely (DCOM) over the network in a distributed objects environment, as illustrated in Fig.1.

2. Automation (OLE automation)

Standard applications, such as word processors and spreadsheets, can be written to expose their internal functions as COM objects, allowing them to be "***automated***" instead of being manually selected from a menu. For example, a script could be written to extract data from a database, summarize it and draw the graphics from a spreadsheet and place the results into a text document.

3. Controls (OLE controls, ActiveX controls)

Applications can invoke COM objects, called "controls," that blend in and become just a part of the program. ActiveX controls can also be downloaded from the Internet to make a Web page perform any kind of processing.

4. Compound Documents and ActiveX Documents

Microsoft's OLE compound documents are based on COM, which lets one document be embedded within or linked to another. ActiveX Documents are extensions to OLE that allow a Web browser, for example, to view not only Web pages, but any kind of document.

5. Programming Interfaces

Increasingly, Microsoft is making its standard programming interfaces conform to the COM object model so that there is continuity between all interfaces.

D. About DCOM

Microsoft's Distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers—on a LAN, a WAN, or even the Internet. With DCOM, an application can be distributed at locations which are required by the customer and the application. Because DCOM is a seamless evolution of COM, one can take advantage of the existing investment in COM-based applications, components, tools, and knowledge to move into the world of standards-based distributed computing. In the process, DCOM handles low-level details of network protocols so one can focus on the real business thus providing great solutions to the customers.

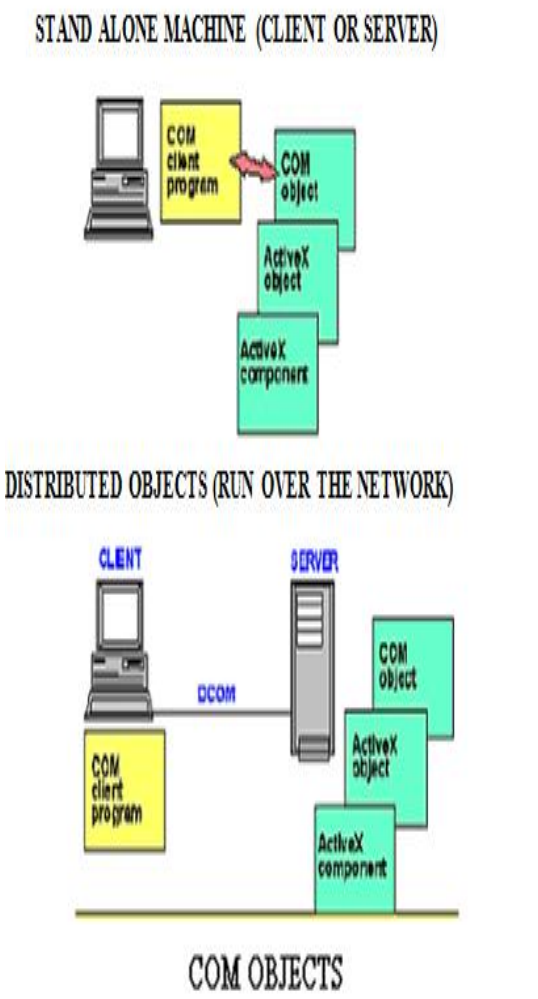

Fig. 1: Interaction of COM objects with client programs.

E. Practical Implementation Of Com Objects

COM objects consist of two types of items, namely properties and methods. Properties are the data members, while the methods are member functions. COM objects completely hide their data and expose their methods through a construct called an interface, as demonstrated in Fig. 2, 3 and 4 respectively. A COM interface is a grouping of related methods that is uniquely identified for all programs and all time (by an Interface ID). Interfaces are used to encapsulate COM object feature sets.

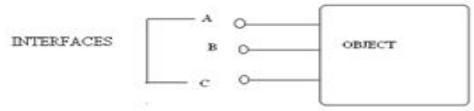

Fig. 2: A component object supporting three interfaces

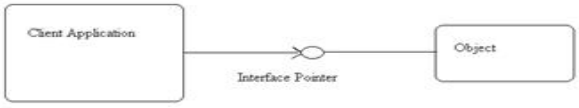

Fig. 3: Interface between client and object

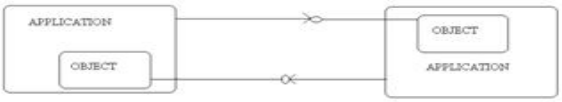

Fig. 4: Interfaces between two applications and two objects

The most fundamental feature of COM is multiplicity of use of COM objects. The IUnknown interface exposes this feature set in several methods (AddRef, Release, and QueryInterface) which determine the common behavior governing all COM object lifetimes and the way in which interfaces on COM objects are acquired. This interface is the main interface for all other interfaces and is the base class from which all other COM interfaces are derived. No matter what they do, all COM objects have to implement the IUnknown interface compulsorily.

Each object implements a vtable (shown in Fig. 5). A vtable is nothing more than an array of pointers to member functions implemented in the object. This vtable is shared between all the instances of the object also maintaining the private data of each object. A client application evokes an instance of the interface and gets a pointer to a pointer that point to the vtable. Each time a new interface to the object is instantiated, the reference count of objects is incremented with AddRef(). Conversely, each time a reference is destroyed, the reference counter is decremented with Release(). Once the reference count is zero, the object can be destroyed. In order to see what interfaces an object supports, one can use QueryInterface().

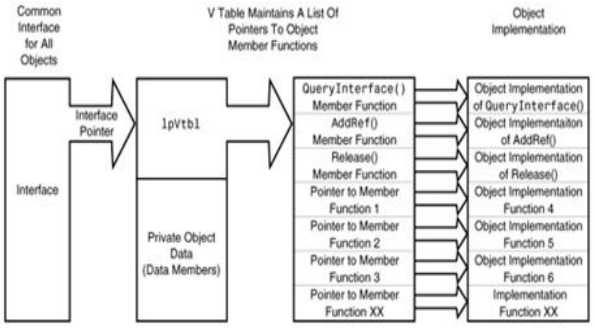

Fig. 5: Representation of a vtable

F. Structured Storage

Most computing platforms today have heterogeneous file systems, making sharing data difficult. In addition, these file systems originated during the mainframe days when only a single application was able to update and in some cases access that data too. COM is built with interoperability and integration between applications on dissimilar platforms in mind. In order to accomplish this, COM needs to have multiple applications write data to the same file on the underlying file system. Structured Storage addresses this need.

Structured Storage is a file system within a file itself. One can visualize it as a hierarchical tree of storages and streams, as shown in Fig. 6. In this tree, each node has one and only one parent, but each node may have from Zero to several children. The folders are the storage nodes, and the files are the streams. Structured Storage provides an organization chart of data within a file. In addition, this organization of data is not limited to files, but includes memory and databases as well.

Stream objects contain data, much like files in a traditional file system. This data can be either native data or data from outside objects. Storage objects are compatible at the binary level; thus, in theory, they are compatible across platforms that support COM and OLE.

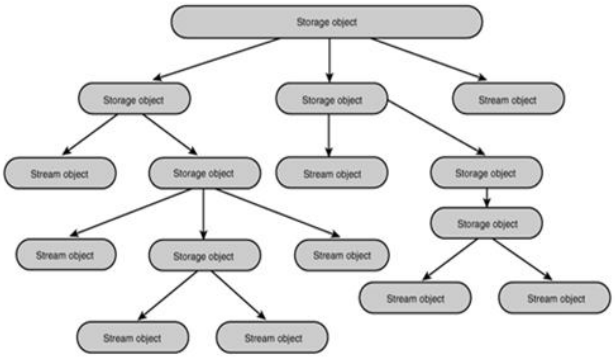

Fig. 6: Hierarchy of storage

F. Monikers (Persistent Naming)

Monikers are a way of referencing a piece of data or object in an object-based system such as COM. The original use for monikers was in OLE linking. When an object is linked, a moniker is stored that knows how to get to that native data. For example, if you link a sound file into a Word document, the WAV file is not stored natively in that document. A moniker is created that can intelligently find the WAV file object. One can think of a moniker as a mapping. However, a moniker is more than just a name—a moniker is a COM object.

G. AUTOMATION: MAKING TASKS EASIER AND QUICKER

Automation basically enables one to manipulate the properties and methods of an application from within another application through the use of high-level macro languages and scripting languages such as VBScript and JavaScript. This enables one to customize objects and provide interoperability between applications.

In the world of Automation (formally known as OLE automation), there are Automation Servers and Automation Controllers. An Automation Server is a component or application that exposes properties and methods for use by other applications. Microsoft Excel is a good example of an Automation Server, because it exposes services that can create and manipulate worksheets, cells, and rows. Various forms of Automation are further elaborated in Fig.7.

Description of the properties and methods that are available through an Automation Server are available in an Interface Definition Language (IDL) file or can be made available as a type library. A type library is a binary representation of the information in an IDL file, although it has slightly lower fidelity.

In Visual C++ (6.0), there is a utility named OLE View that reads and graphically displays the contents of type libraries. One can use this utility to display the properties and methods exposed by Automation Servers.

Automation Controllers are client applications which use the properties and methods exposed by Automation Servers. Automation Controllers work through an interface called IDispatch. All interfaces that support Automation are derived from IDispatch.

There are three basic ways in which we can use Automation: MFC, #import, and C/C++:

1. With MFC, Visual C++ ClassWizard is used to generate "wrapper classes" from the Microsoft Office type libraries. These classes, as well as other MFC classes, such as COleVariant, COleSafeArray, COleException, simplify the tasks of Automation. This method is usually recommended over the others, and most of the Microsoft Knowledge Base examples use MFC.

2. In the project demonstrated, this method has been used to automate Microsoft excel.

3. #import, a new directive that became available with Visual C++ 5.0, creates VC++ "smart pointers" from a specified type library. It is very powerful, but often not recommended because of reference-counting problems that typically occur when used with the Microsoft Office applications.

4. C/C++ Automation is more difficult, but sometimes necessary to avoid overhead with MFC, or problems with #import. Basically, we work with such APIs as CoCreateInstance(), and COM interfaces such as IDispatch and IUnknown.

It is important to note that there are slight differences between Automation from C++ compared to conventional C, because COM was built around the C++ classes.

Thus, automation enables tasks that are normally selected from menus to be automatically executed. For example, a small script could be written to extract data from a database, put it into a spreadsheet, summarize and chart it, all without Manual intervention.

Virtually any internal routine can be written as a COM object and its interfaces exposed to other programs. Microsoft applications such as Word and Excel are written as COM objects, and they not only allow their functions to be automated, but offer programmers a toolbox of functions that can save them the time and effort of writing similar routines.

The essence of automation lies in the fact that stand-alone application like Excel has many mathematical, statistical, charts and financial functions. Other applications can take advantage of these functions simply by calling Excel through the program. The end user is oblivious of the fact that Excel is being used.

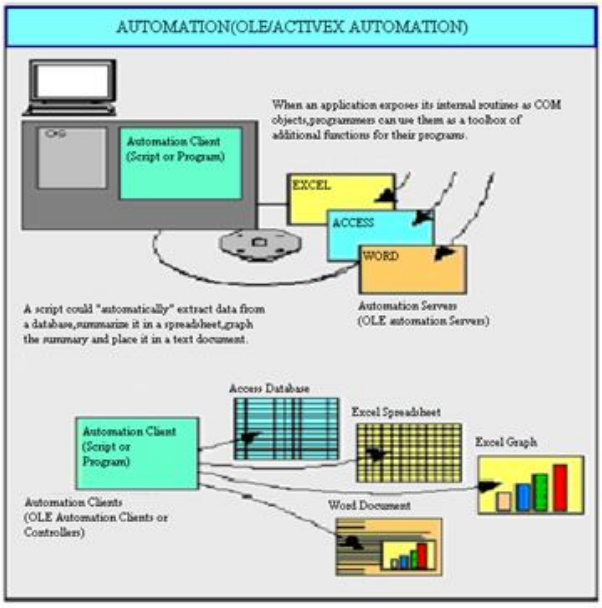

Fig. 7: Forms of Automation

II.IMMEDIATE APPLICATION IN DEFENCE (R&D)

This model is being used for data analysis, visualization and storage of wide range of materials having direct defence application. The model helps in segregating, grouping and extraction of information pertaining to defence materials especially in structural, stealth and polymers application. The model further aids in better visualization of the data pertaining to these materials. Data storage, visualization and portability would be very cumbersome using conventional development environment like C++, Java etc.

The scheme we have created uses the class wrappers generated from EXCEL 2002 object library .The application is created using MFC(Microsoft foundation class) and excel type library. It is a generalized version of automating component integration with COM compliant application such as the Microsoft Office Applications.

The objective here is to automate MS EXCEL 2002. The application prompts the user to enter parameters for 2 variables and then generate values for the variables based on the input by the user. The dialog box created is shown in Fig.8. It then creates a pie chart, line chart, bar chart or a XY scattered chart based on the choice of the user, as shown in Fig. 9 and Fig. 10.

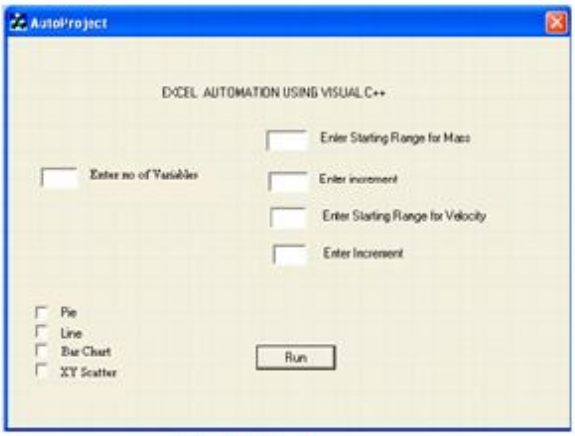

Fig. 8: Dialog Box

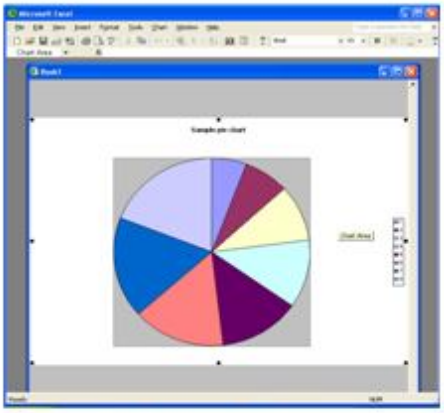

Fig. 9: Sample Pie Chart

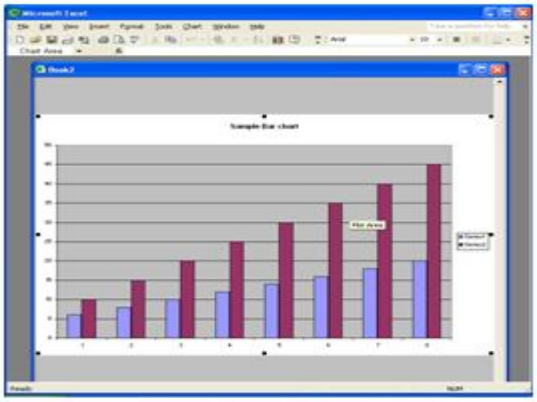

Fig. 10: Sample Bar Chart

2.1 Development Tools and Environment

MFC Application Wizard (Appwizard) is used to create the dialog based application. Idispatch interfaces and member functions defined in the excel.exe type library are used. The Generated class wrappers are derived from coledispatchdriver, And The appropriate declaration header file is named excel.h. additional codes and functionalities were added to load and enable com services library. The project was created in microsoft visual c++ 6.0 environment.

2.2 Significance of Idispatch Interface in the Context of Project

Ole automation controllers use the ole idispatch interface to gain access to objects that implement this interface. Idispatch keeps track of object members (methods and properties) by the dispatch id(dispid). Before an ole automation controller can find a method or a property, It must have a dispid that maps to the name of the mumber. Then it calls idispatch::invoke to locate property or invoke the method, packaging the parameters for property or methods into one of idispatch::invoke parameters. At run time, controllers get dispids by calling the idispatch::getidofnames function. this is called late binding because controller binds to the property or method at run time.

Here, Since Object is described in a type library, An Ole Automation Controller reads the dispids from the type library at the compile time. This is called id binding. call to getidofnames is avoided. Since it requires only one call to idispatch rather than 2 calls required by late binding, id binding is generally about twice as fast. Id binding is advantageous as performance is better since no binding are required at run time, So we get better performance in terms of compile time. Over and above the user can receive more information like error messages etc.

III.THE ADVANTAGES AND DISADVANTAGES OF USING THE COM TECHNOLOGY

3.1 The main advantages of the COM technology are:

A. The need to build and update entire application every time is eliminated. Rebuilding a single component is enough.

B. It promotes component based software development which has several advantages like ability to use pre-packaged components and tools from third party vendors into an application and support for code reusability in other parts of the same application.

C. COM makes it possible for different language components, which adhere to COM specifications to interact with each other.

D. It helps to access components loaded in different machines on the network.

E. It can segregate applications via binary firewalls or interfaces that can reduce or eliminate dependencies between primary elements.

F. It provides many features to developers in industry like language independency, binary standard, wide software industry support etc.

G. Object Oriented Features

All the COM objects are Object Oriented as its name implies. There are three basic Object Oriented concepts:

a. **Encapsulation**: This is implemented by the interfaces provided by the COM Objects. The interface hides the implementation details from the end user and provides the functionality to the user.

b. **Polymorphism**: This is often referred as "One Method Multiple Interfaces". A COM object can define a single method to perform a specific operation; but that operation could be implemented through various ways.

c. **Inheritance**: When the user wants to incorporate some additional functionality to an existing COM object, he can enhance the existing COM object by inheriting a new COM object from it.

H. Loose Coupling

In software, which uses COM objects, one can easily replace an existing COM object with another COM object written in entirely another language as long as the signatures of the methods in both the COM objects remain same. In such a case, there will not be any change in the existing software code that uses the COM object.

I. Binary Language

Since most of the COM libraries are in binary language, it could be used by any application written in any language. So COM is language independent.

J. Resource utilization

Every COM object will be destroyed automatically as long as no client is using that object actively. This is implemented by COM using a technique called reference counting. Every COM object will maintain a reference count (the number of clients using that COM object); Once that count reaches zero (that means no clients are actively using that COM object), then that COM object will be destroyed automatically. With this approach, we can increase resource utilization in a single application. Resources such as memory will be best utilized by releasing the inactive/unused COM objects.

3.2 Specific Drawbacks Of Com Objects

A. Difficult COM Component Development

COM components are often difficult for developers to implement. An extensive amount of code must be provided to implement a valid COM object. Furthermore, the software debugging tools for COM objects are less developed than their non-component counterparts. Since COM objects can be active on multiple computers, it is often difficult to determine the source of malfunction(s), if any.

B. Divergence in Standards

The COM architecture only specifies how software components communicate – not the specific interfaces they should use. Microsoft has already defined many interfaces that provide standard features such as drag-and-drop, data transfer, and persistent storage. These are not the only valid interfaces; any developer can define and implement their own custom built interface. Although this in one of the greatest strengths of COM, it is also one of its greatest weaknesses. With unchecked interface proliferation, developers will create their own closed set of interface standards that will be incompatible with the work of other developers. This problem has been partially averted through Microsoft's use of a standard scripting interface referred to as "IDispatch." The coordinated development of interface standards would be necessary for the ongoing success of COM.

C. The other major drawbacks include difficulty in integrating internet technologies and expensive, difficult, and undependable deployment.

IV. ADVANCED COM TECHNOLOGIES

4.1 Microsoft Transaction Server

Microsoft Transaction Server (MTS) is a component-based transaction processing system for building, deploying, and administering robust Internet and Intranet server applications. In addition, MTS allows one to deploy and administer your MTS server applications with a rich graphical tool (MTS Explorer). MTS provides the following features:

1. The MTS run-time environment.
2. The MTS Explorer, a graphical user interface for deploying and managing application components.
3. Application programming interfaces and *resource dispensers* for making applications scalable and robust. Resource dispensers are services that manage non-durable shared state on behalf of the application components within a process.

The three-tier programming model provides an opportunity for developers and administrators to move beyond the constraints of two-tier client/server applications. It also emphasizes a logical architecture for applications, rather than a physical one. Any service can invoke another service and can reside anywhere.

4.2 COM+

It is the enhancement of the Microsoft Component Object Model (COM) that enables programmers to develop COM objects with more ease.

For example, COM+ allows native C++ calls to be translated into equivalent COM calls. In addition, instead of defining COM interfaces in the traditional IDL language, they can be defined by more familiar programming syntax.

Where the Component Object Model (COM) ends, COM+ starts. COM+ extends COM to let you create components that scale better for:

- **Security in Distributed environments**: COM+ makes it easier to build components that can utilize the security subtleties of Distributed COM (DCOM) .
- **Component Services**: For event notifications, synchronization, and de-coupling of event sender and receiver (queued components, etc.).
- **Performance in Distributed environments**: COM+ makes it easier to build components that can stand the stress of high-volume instantiation requests over short periods of time.
- **Deployment in Distributed environments**: COM+ makes it easier to distribute server components or client-side components (typelibs and proxy-stub DLLs) without having to copy, unpack, register and configure each of them.

V.CONCLUSIONS

The project has been executed and tested successfully. The incorporation of any modifications in the project is very easy as only minor changes in the main handler code would be needed. The automation makes certain redundant tasks much easier and quicker to handle. Such applications are extremely useful for certain users such as data analysts who have to regularly analyze and summarize data. The application can be easily modified to read data from a database, extract it to a spreadsheet and generate suitable chart based on the data, or to prepare a text report based on it.

Thus, it may be concluded that the COM technology is truly versatile and extremely advantageous. Nowadays, this technology is gaining popularity as most of the languages support development of COM objects. The activities of an operating system and applications would be strongly COM oriented in the coming future. This would surely make them more amicable to the users, even to a novice.

ACKNOWLEGMENT

Authors are thankful to Director DMSRDE, Kanpur who has provided all the support needed to satisfactory perform the task.

REFERENCES:

The following books and web resources have proved to be extremely useful in preparing the project and the paper.

BOOKS

[1] Visual C++ 6 unleashed-Mickey Williams and David Bennett

[2] COM/DCOM Unleashed – Daniel Wygant

[3] Inside COM(Programming Series) – Dale Rogerson

[4] Professional COM Applications with ATL – Sing Li and Panos Economopoulos

[5] Inside Distributed COM(Mps) – Guy Eddon, Henry Eddon

[6] Essential COM(The Addison-Wesley Object Technology Series) by Don Box

ONLINE RESOURCES

[1] MSDN LIBRARY VISUAL STUDIO 6.0

[2] http://www.microsoft.com/technet

[3] http://computing-dictionary.thefreedictionary.com (Definitions and Figures)

[4] http://www.howtodothings.com/computers/a1239-attributesadvantages-of-com.html

[5] http://www.peterindia.net/COMOverview.html

[6] http://www.richardhaleshawgroup.com

[7] http://media.wiley.com/product_data/excerpt/23/07645599/0764559923.pdf

[8] http://www.emory.edu/BUSINESS/et/com/

[9] http://en.wikipedia.org/wiki/Object_linking_and_embedding (OLE references)