

Case study of DCOM- A Technology To Support Real-Time Embedded Systems

Pritam Surendra Kokate
 Department of Electrical and Computer Engineering
 Mississippi State University
 psk8@ece.msstate.edu

ABSTRACT

Real-time Embedded Systems is constantly growing research field in these days and is achieving a great importance. Research is carried to make everything controlled in our normal life through any web applications. Embedded systems play a major role in making it possible. Every small device or an application needs to be connected to a proper source to activate it remotely. To make this possible the system should be capable of interacting with hardware as well as with software at the same time. This requires a middleware that can transfer actions to all the components in the system. Distributed Component Object Model (DCOM) is one of such a middleware. DCOM is language independent and location transparent and thus, can be an ideal middleware for an embedded system.

Index Terms – *DCOM, component, location transparent, language independent, class id*

I. INTRODUCTION

To make a real-time embedded system, interaction between various technologies and their components plays a very important role. The system should be capable to handling clients from different locations. The clients can also be software or hardware. They should have a common media of communicating between each other. To make a new system right from the beginning is a very tedious job. The system should be capable of reusing elements that are used in some other systems. Various systems need to interact with other heterogeneous systems. To have a common platform or an interaction between two systems a middleware is needed. There are various middleware available that interacts between different objects such as Common Object Request Broker Architecture (CORBA), DCOM etc. This paper mainly discusses pros and cons of DCOM architecture and its applications in embedded systems.

Section 2 discusses primitive methods of coding. This section further discusses the advantages and disadvantages of such methods with respect to the feasibility of real-time embedded systems. Section 3 discusses the basic fundamental of component in a Component Object Model (COM). Section 4 describes the architecture of a simple COM model and how it is distributed over the network to become a DCOM. This

section also covers the difference between a COM model and a DCOM model. Section 5 answers the question as to “why a DCOM model should be used?” This section also highlights the actors/creators in the DCOM. Few applications of DCOM are also included in this section. Section 6 gives the conclusions and ideas for future refinements and applications of these results.

II. PROBLEMS IN CONVENTIONAL PROGRAMMING

In any software language, especially the very familiar object oriented language (OOPS); the source code is reused. Reusability is very important in any application. The usage of any application is determined by its reusability. The components of the software should be easy to use, easy to update and should be independent of language and compilers [1]. In conventional programming languages such as C++, Java whenever a class is designed it is usually reused by its children, if inherited, or defined by another class if it is an abstract class. In such code inheritance, the entire source code of the original class is used. To be more precise, consider an example in Unified Modeling Language (UML) shown in fig 1.

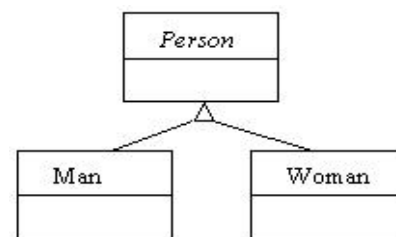


Figure 1. Source-code instantiation

Consider a parent class “person” and two other classes “man” and “woman” that inherits this class. If any of the child class e.g. man, is instantiated the entire source code of the parent class *person* is instantiated as well. This instantiation compiles the parent class whenever a new child object is created. This introduced the reusability but eventually the source-code was instantiated and used. Thus if a source-code is reused the code must be recompiled.

The advantage of this method is

- Best for small-size code

The disadvantages of this method are

- More system resources used for large sized code
- Tough for complex systems
- Code needs to be changed for a new compiler

To avoid system resources another method was introduced instead of source-code reuse and that was using static linking. This static link is the link between the component that is in the binary format and the application. But, even this method did not help reducing system resources to a great extent since to use this method a library of source code must be compiled to binary form. Thus using a vast library without using entire functions of the library consumed almost the same system resources as much as in source-code reusability. The link is known as a static link because it is done during compile time [1].

There is another way of linking and that is done at run-time, which is also known as dynamic linking between the application and the pre-compiled library. But the disadvantages for this method remain the same, as this either does not solve the problem of language dependency.

III. WHAT IS A COMPONENT

“A component is a reusable piece of software in binary form that can be plugged into other components from other vendors with relatively little effort” [3]. Application developers create software components to enable the reusability of the codes. An element known as the Dynamic Link Library (DLL) is used to link function/methods with any application. A single application cannot use two DLL’s exporting the same function. But instead using a COM solves the problem. Different applications using different COM servers with identical interface and can be on same workstation. This principle when used over a distributed network or even on client-server applications is called as DCOM.

While designing a new distributed system high cohesion between the components should be maintained. High cohesion means all the components interacting more amongst each other should be placed closer to each other [3]. The bandwidth can be fairly increased by using high cohesion [4]. Use of DCOM completely hides the location of the component. It does not matter if the component is in the same workstation or the method is being used in different workstation. Just a simple change in the configuration changes the component connectivity to each other. There is no need for a change in the source code or any recompilation when DCOM is used.

IV. DCOM ARCHITECTURE

To overcome the drawbacks from Section II and delving into a new concept from Section IV a new technique was followed known as the pointer technique. A function table is developed to resolve the language and the compiler dependencies. This was the basic principle of COM. There is a pointer reference to the component from the client, which is

also shown in Fig 2. This figure shows client and component are in the same process.



Figure 2. COM used in same process.

The pointer reference is made to the windows registry where the binary of the component is stored. The binary of this component is stored in a function table and is identified by a class-id number (CLSID). When this principle is used over a distributed network or even like a client-server applications is called as DCOM. While communicating between different components another issue that has to be taken care of, is security. Almost all the operating systems have their process shielded by providing proper security level, e.g. Kernel level. A client may not be able to directly communicate with component of another process of a higher security. DCOM provides a network protocol by intercepting calls from client and forwards to the component in another process. The architecture of DCOM is shown in Fig 3.

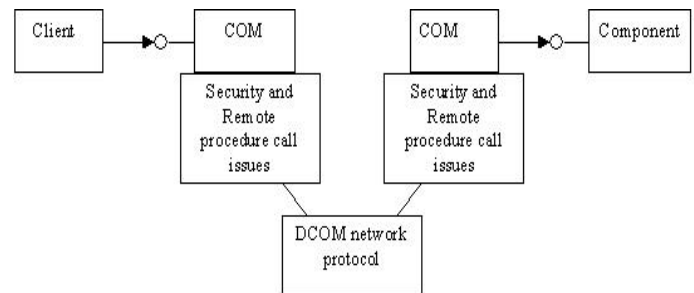


Figure 3. DCOM architecture.

The architecture uses the Remote Procedure Call (RPC) to generate standard network packets. There is an inbuilt counter attached to DCOM that counts the number of clients attached to a component. If any component is detached the counter decreases. The architecture for COM clients on different process remains the same except for the network protocol a local interprocess communication is used [2].

V. WHY USE DCOM?

Designing any distributed application is complex and is a whole new kind of design in itself. The utilization of such a complex system should be high enough to design from the beginning. DCOM increases interoperability by using binary and network standard. Providing a binary and network standard makes an application to be used by different clients. These clients can be on same workstation or may retrieve information from a server in the network. Thus the clients need not recompile the binary objects. COM objects can have many interfaces unlike objects in CORBA [5].

The scalability of the application can be increased by using distributed applications thus optimizes the use of network and

computer resources. This property of DCOM is also known as location transparency. DCOM also gives a language independence, which means a component can be written in any language can be utilized using DCOM.

Almost any language can be used to create COM components. Thus creators can be those languages that make the DCOM components and the players are those who use these components. The actors and players can be the same language too. DCOM can interact with Java, Microsoft Visual C++, Microsoft Visual Basic, Delphi, PowerBuilder, and Micro Focus COBOL. Since DCOM is language independent, the components can be made in any higher languages such as VC++ and can be re-implemented in C++ or Java.

Some other applications of distributed computing can be multiuser games, teleconferencing applications and chat. In client-server applications, many times different components run on different machines and they need to interact and communicate between each other. DCOM benefits in running correct applications/components in right places and also optimizes the computer resources and network utilities [2]. With the use of DCOM load on a single application can be divided into parts and can be made work together, thus increasing the throughput of the entire system.

DCOM is directly available with Win-98 or Win-CE packages. As an embedded systems application DCOM is also used in server of Pocket PC on a Windows-CE operating system [6].

The DCOM architecture has a few disadvantages [7] as shown below

- Not a true open standard.
- Behaves poorly in disconnected environment
- Complex to program in C++
- Does not work well through fire-walls

VI. CONCLUSION

We have just seen the architecture and technology that supports DCOM. In embedded systems we need to have a middleware that can communicate between various objects or components. The components can be either of hardware or of software depending on the need of the application or the system. DCOM should solve the purpose for most of the applications. The architecture of DCOM is language independent and location transparent. There are few applications towards embedded systems and just using DCOM. DCOM is used with CORBA, which is again an object request broker and is platform independent too. Thus in future DCOM seems to have a good application in the field of Embedded Systems.

VII. ACKNOWLEDGMENT

The author is thankful for Dr. J.W. Bruce for his support and also suggesting such a good research topic in the field of embedded systems.

VIII. REFERENCES

- [1] Using COM for embedded systems; S. Malliet. Embedded Systems Conference, Session 216
- [2] MSDN library for DCOM http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp.
- [3] MSDN SPECS for COM <http://www.microsoft.com/COM/resources/COM1598B.ZIP>.
- [4] Applying UML Patterns- Craig Larman.
- [5] Using CORBA to Accelerate Distributed Application Development and Improve Network Management; Jon Currey, Embedded Systems Conference, Session 242.
- [6] http://msdn.microsoft.com/chats/embedded/embedded_032502.asp.
- [7] Achieving Reliable Distributed Object Communications within Unreliable Networks; Mike Preradovic, Intrinsyc Software Inc, Embedded Systems Conference, Session 411.