Analysis of Component Object Model and Common Object Request Broker Architecture

Abhishek Maheshwari, Sharnil Pandya, Aishwary Rawat

Abstract— Component Object Model and Common Object Request Broker architecture enables software components to communicate with each other. In this paper, the various analytical aspects of COM and CORBA are explored. The similarities, differences and application in the real world are seen. The main objective of this paper is to measure the performance of both bridges in different bridging configuration as specified in the Interworking Specification.

Index Terms— Object broker request, Interface Definition Language, Stubs, Marshalling, Demarshalling, Client, Server.

1 INTRODUCTION

COM, Short for Component Object Model, was introduced by Microsoft. It enables software components to communicate with each other. Hence the development of the software becomes easier as it is built from parts and not started from scratch. [1]
The interface provides a logical grouping of services. [2] In COM, an interface is defined as a memory structure containing an array of function pointers. Hence, the term 'Binary' is also used to describe COM and this property makes COM language independent as long as the language is able to access and define this type of function pointer structure. COM interface is implemented in such a way that they are linked dynamically. This makes swapping of components possible. DLL (Dynamic Link Library) file or EXE(Executable) file are there to package the components which are necessary to allow communication between different components.

CORBA, short for Common Object Request Broker Architecture, was introduced by OMG (Object Management Group), automates various network programming tasks such as object registration, location, and activation; framing and error handling; request demultiplexing; parameter marshalling and demarshalling; and operation dispatching. CORBA uses the ORB architecture (Object Request Broker) which provides a mechanism for transparently communicating client requests to target object implementations. [3] The ORB decouples the client from the details of the method invocations. This makes client request appear to be local procedural calls.

2 VARIOUS ASPECTS OF COM AND CORBA

There are many similarities and differences between COM and CORBA. The strengths, weaknesses and the features that are included in both. These aspects are across various fields like interface, data types, Proxies, Stubs and Skeletons, marshalling and demarshalling and Objects:

2.1 Similarities

There are various similar properties that are included in both, COM and CORBA.

COM and CORBA uses their own IDL (Interface Definition Language) to describe interfaces. IDL is a specification language used to describe a software component's interface. It describes an interface in a language-independent way, enabling communication between software components that do not share a language. In case of data types, both supports a rich set of data types. They also support constants, enumerated types, structures and arrays.[4]

CORBA and COM rely on client stubs and server stubs to handle remote issues. Stubs stands for a client side object participating in the distributed object communication. COM and CORBA generate different client stubs and server stubs from IDL. The stub acts as a gateway for client side objects and all outgoings requests to sever side objects that are routed through it. [5]

They handle marshalling in client stubs and sever stubs. Users do not need to worry about marshalling. Marshalling is the process of transforming the memory representation of an object to a data format suitable for storage and transmission, and it is typically used when the data must be moved between different parts of the computer program or from one program to another. [6]

Object creation, object invocation and object destruction are some of the features which are similar in both COM and CORBA. Both use factories to create object instances. Both allow for method invocation similar to native environment method invocation. COM and CORBA rely on reference counting to determine when an object can be destroyed. [4]

2.2 Differences in COM and CORBA

CORBA IDL, in terms of interface, is simpler and elegant. COM has better tool for support for creating and managing IDL than CORBA. An interface defines a set of methods that an object can support, without dictating anything about the implementation. The interface marks a clear boundary between code that calls a method and the code that implements the method. [4]

COM has automation types. Automation compatible interfaces are supported in more client environments than non-compatible interfaces. Because the non-compatible interfaces are not guaranteed to work other than C++. Any CORBA interface can be used from any CORBA client. [4]

COM client & server stubs are called as Proxy & Stub and in CORBA called as Stub & Skeleton. COM proxy-stub DLLs are used by all language environments. In CORBA, a separate stub-skeleton must be generated for each ORB/language. COM allows automation-compatible interfaces to use type library marshaling, thus eliminating the need for customized stubs.

COM calls object handles as interface pointers and CORBA calls as object references. CORBA supports multiple inheritance in the interface hierarchy. COM supports single inheritance only; however a COM object supports more than one distinct interface. [4]

COM has a standard factory interface called IClassFactory CORBA factories are customized persistent CORBA objects. COM's error-handling mechanism is based on HRESULT return values. CORBA supports user-defined exception types in IDL. COM supports distributed reference counting and garbage collection. CORBA reference counts are maintained separately in the client and server [4].

2.3 Strengths of CORBA and COM

CORBA strongly supports the Unix and mainframe systems. It's a cross platform and a multi-vendor architecture which is of an Industrial Standard. Excellent version of implementation are available in the market. CORBA also binds a wide range of programming languages. [7] The main strength of CORBA architecture is its simpler programming interface. All the objects or interfaces can be called dynamically at run time through a data driven interface: CORBA DII. Dynamic Invocation Interface is an API which allows dynamic construction of CORBA object invocations. It is used at compile time when a client does not have knowledge about the object it wants to invoke. [7] With this interface, an argument list is marshalled, a function is named, and a request for service is sent to the object server. DII will usually have asynchronous mode of operation. CORBA also supports multiple inheritance of interfaces. IDL supports multiple inheritance of interfaces. A (slightly contrived) example is:

```
// IDL
// In for example, "bank.idl".
// A bank account.
interface Account {
        readonly attribute float balance;

        void makeDeposit(in float f);
        void makeWithdrawal(in float f);
};

// Derived from interface Account.
interface CheckingAccount : Account {
readonly attribute float overdraftLimit;
};

// Derived from interface Account.
interface DepositAccount : Account {
};

// Indirectly derived from interface Account.
interface PremiumAccount :
                CheckingAccount, DepositAccount {
}; [8]
```

COM has strong versioning support of interfaces; one can easily support upward or backward compatible interfaces on an object. Good support for fine-grained objects, with in-process activation and no requirement for persistence support. COM also separates "class" from "interface" -- each object/class/instance will normally support multiple interfaces, and it's easy to switch between them. Microsoft backs it. [7] They have lots of money, and widely used tools (on Windows platforms). MS is now encouraging use of .NET/SOAP, but COM is still supported. Tool support (like within VB, VC++, J++) -- but only with Microsoft tools on the Windows platform.

COM has more flexible pointers; CORBA object references can only be to whole objects (as in Java), whereas COM pointers can point into the middle of structures (as in C++). It also has strong definition of object identity: COM has a clearly-defined way to determine if two different interface pointers really refer to the same object; even if the two interfaces aren't related to each other in any way by inheritance. [7]

COM gives a better Separation of Concerns. COM components can be used locally without incurring the overhead of distribution or ORBs.

COM and CORBA has some features supported by them both. Support for Reflection is optional in both. COM's TypeLibrary cannot encode everything that can be expressed in MIDL.

Actual support for asynchronous processing is weak or absent in both. CORBA provides "one way" operations that are so unspecified as to be practically useless (they are not guaranteed to be reliable, or asynchronous), but does allow asynchronous calls of synchronous methods through the DII.

This was true a few years ago, but no longer. Recent versions of the CORBA specification provide very rich asynchronous invocation mechanisms. And COM+ has support for asynchronous invocation as well.

3 Conclusion

Based on the analytical comparison made between COM and

CORBA, it can be seen that CORBA architecture has certain problems associated with it that are solved using COM. The language, platform independent and suitability for all the distributed systems led to a lot of complexity. Language mapping is considered as "Unnatural" and there is no inheritance for exceptions.

ACKNOWLEDGMENT

We wish to thank Nirma University for its continuous support that helped us to do a research in this area.

REFERENCES

[1] Wikipedia- Component Object Model.
[2] ObjectBridge COM/CORBA Enterprise Client User's Guide.
[3] Wikipedia- Common Object Request Broker Architecture.
[4] Middleware Technologies [MCA II yr, Anna University] by Antony Arnold.
[5] Wikipedia- Class Stub.
[6] Wikipedia- Marshalling (computer Science)
[7] Wiki- COM VS CORBA
[8] IONA Technologies- Inheritence chapter 13.