**A com compliant object has many interesting traits.**

1. **COM is language independent:**

A com interface can be written in C++, COBOL, vb, java etc. The only true requirement that a language must satisfy is that it should be able to generate a VTABLE (binary layout) mapping to a COM object.

1. Not only Developers can write COM objects in any language, the COM objects themselves can be accessed by any COM language.

For example a JAVA application can access a COM component written in C++ or Delphi.

1. COM encapsulates the inner implementation details of the object including the

Language it was written in. All that the COM object user sees is a set of well-defined interfaces supported by the object.

1. **COM provides location transparency**

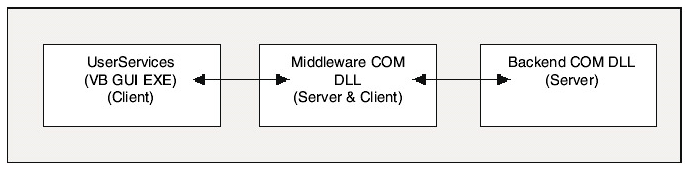
Before we understand location transparency lets add a few words to the COM vocabulary (lexicon)

1**. Process:** A process is a partition of memory containing a running application (i.e. an active main thread) along with any necessary system resources and subsidiary threads and external binaries required by the application.

2. **Client:** A client in COM is any piece of software that makes use of a COM object.

**3. Server:** In COM we define a server as a binary package (DLL or EXE) that contains one or more COM objects. Typically a single com server is known to be a home of many related COM objects, each supporting many number of interfaces.

Always keep in mind that the distinction between client and server can become blurred a little bit as it is common to have a client using a server that is a client to another server. The following diagram explains it:



Now with the three terms in place i.e. Client, Server and Process we can now distinguish between the possible relationships a client and server may have

1. An In-Process Relationship.

2. An Out of Process Relationship.

3. A Remote Relationship.

**UNDERSTANDING THE IN-PROCESS RELATIONSHIP:**

The relationship is abbreviated as in-proc relationship. In-proc servers are loaded into the same **memory partition (process)** in which the client application is running. They are in the same memory partition of the client that they are servicing.

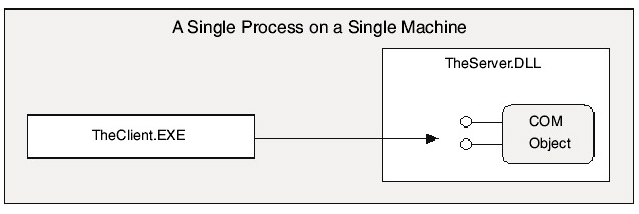
**Key Benefit:**

Speed. Since the servers are loaded into the same memory partition as client, the calls to the interfaces are as fast as making a local function call as the client and the server are **communicating through interface pointers.**

**Drawback:**

1. **Fault Tolerance**: The biggest drawback of an in-proc server is robustness or low level of fault tolerance. If the server crashes it will bring down the entire client process too.

2. **Security**: The in-process COM server will always take the security context of the Client Application they are loaded into.



**NOTE:**

COM has different threading models. IF the client and the server share different threading models then direct interface connection is not achieved.

**UNDERSTANDING THE OUT-OF-PROCESS RELATIONSHIP**

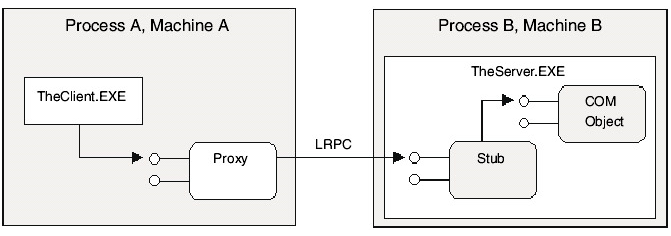
The out-of-process relationship is also termed as a local relationship. The client and the server reside on the same physical machine but they run in their own individual (distinct) memory segment as separate processes.

The two processes run with their own security contexts hence a higher level of security.

The major drawback is that it is slower as the requests have to be packaged, sent and unpackaged between process boundaries. While COM takes care of this packing and unpacking using a technique called **universal marshalling** and we rarely have to take care of it, but the process itself is time consuming.

**The major advantage** is that the system is fault tolerant as if the out of process com server crashes the client keeps running in its own process space.

The following diagram shows the out-of-process COM server and the relation between client/server.



As shown in the above example the client and the server processes communicate and send information through **LRPC (Lightweight remote procedure call).**They run on the same machine but in different processes.This is a proprietary communication protocol under the hood of the COM runtime.

In this local relationship the client and the server do not communicate directly through interface pointers but communicate using stubs and proxies. The data is transmitted between stubs and proxies.

As far as the client is concerned, the proxy is the COM object and can program against it as if it was an in-proc server. This will be discussed further.

**UNDERSTANDING THE REMOTE RELATIONSHIP**

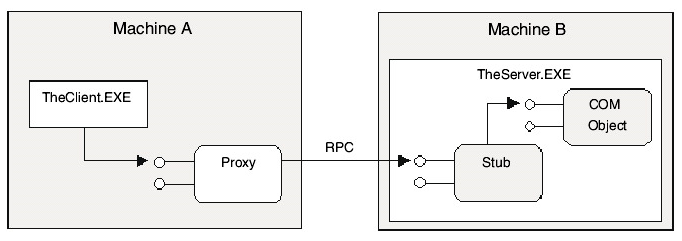
The remote COM server resides physically on any remote machine other than the machine on which the client is running.

This is a core requirement of any enterprise based solution, where the user services, business rules, and data retrieval objects can be distributed and reside on many networked computers on the network.

The remote client server COM relationships are the slowest amongst all the relationships as not only we need to package/unpackaged and marshal (assemble arrange in order) data but we also have to take into consideration the network and latency issues (Bandwidth etc.).

Remote communication is achieved through **DCOM protocol** which uses **Remote Procedure Call** mechanisms to **establish communications and forward information** between the Client and the server.

The following diagram represents a remote relationship:



**NOTE:**

The remote COM servers can be housed in DLLs or EXEs in the remote machine. Legacy COM servers can also be used and remoted using the special COM Surrogate process/utility called the dllhost.exe.

EXE server types were a “Classic” way to remote COM servers especially over LAN. This is still an effective solution.

However these days MTS is more popular. MTS is Microsoft transaction server which is a super surrogate and will host “only” COM dlls while taking care of all the details like threading concurrency, atomic transactions, and memory management and so on.

**LOCATION TRANSPARENCY**

One benefit of com states that COM provides location transparency where the client programs against the proxies as if it was making in-process function calls without knowing the location of the server.

This also states that the client side code used to access the interfaces of the server side COM object need not change based on the location of the COM object.

**Stubs and proxies** provide a level of encapsulation and the COM client treats the proxy object as a real object and can program against it as if the object was running in the clients own process space.

Likewise the COM server believes that the **STUB is the real client** leaving the server code intact.

The COM library does provide a number of methods and data structures to help optimize remote access. But these are optional. Using configuration utilities such as dcomcnfg.exe, you may redirect your client to access a server on any machine on the network without altering a single line of code.

These utilities and library extensions provide a far cleaner approach than we had before location transparency. At that time, developers needed to use different (and unrelated) APIs to move information between in-proc, local, and remote servers. As a COM developer, you do not write different code bases for client access to remote objects, local objects, or in-proc objects. Clients just see interfaces, and location transparency allows that client to be blissfully unaware of the exact location of the server.

**COM IS OBJECT ORIENTED**

Some people argue that COM is not object oriented. But it is.

**ENCAPSULATION:**

COM does a good work in information hiding. All that is exposed to the clients are proxies. The underlying implementation is absolutely hidden from the clients.

**POLYMORPHISM:**

As we have seen from the SHAPES example any object is allowed to implement the same interface with its own unique implementation. COM provides an ad-hoc polymorphism using the same interface.

**INHERITANCE:**

COM does not provide a classical inheritance of binary objects. In other words we cannot say that COM object A derives from COM object B.

(For example, C3DRect used interface inheritance to bring in the definitions of IDraw and IShapeEdit). What we cannot do with COM is something along the lines of the following:

// this is not possible in COM.

// assume *BLOB* is a new keyword that defines a new binary COM object.

BLOB MyServer.NewBlob.EXE: public MyOtherServer.OldBlob.DLL

{

...

};

**COM CONTAINMENT AND AGGREGATION**