.**C.O.M. (Component Object Model)**

COM is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents) and ActiveX (Internet-enabled components) technologies.

COM is an interface technology defined and implemented as standard only on [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) and Apple's [Core Foundation](https://en.wikipedia.org/wiki/Core_Foundation) 1.3 and later plug-in API,[[1]](file:///D:\software1\Component%20Object%20Model%20-%20Wikipedia,%20the%20free%20encyclopedia.html#cite_note-1) that in any case implement only a subset of the whole COM interface.[[2]](file:///D:\software1\Component%20Object%20Model%20-%20Wikipedia,%20the%20free%20encyclopedia.html#cite_note-COM_on_MacOS-2) For some applications, COM has been replaced at least to some extent by the [Microsoft .NET](https://en.wikipedia.org/wiki/.NET_Framework) framework, and support for [Web Services](https://en.wikipedia.org/wiki/Web_Services) through the [Windows Communication Foundation](https://en.wikipedia.org/wiki/Windows_Communication_Foundation) (WCF). However, COM objects can be used with all .NET languages through .NET [COM Interop](https://en.wikipedia.org/wiki/COM_Interop). Networked DCOM uses binary [proprietary formats](https://en.wikipedia.org/wiki/Proprietary_format), while WCF encourages the use of [XML](https://en.wikipedia.org/wiki/XML).

COM provides a stable [ABI](https://en.wikipedia.org/wiki/Application_binary_interface) that does not change between compiler releases.[[3]](file:///D:\software1\Component%20Object%20Model%20-%20Wikipedia,%20the%20free%20encyclopedia.html#cite_note-3) This makes COM interfaces attractive for object-oriented C++ libraries that are to be used by clients compiled using different compiler versions.

COM objects can be created with a variety of programming languages. Object-oriented languages, such as C++, provide programming mechanisms that simplify the implementation of COM objects. These objects can be within a single process, in other processes, even on remote computers.

**E.J.B (Enterprise Java Beans)**

EJB stands for Enterprise Java Beans. EJB is an essential part of a J2EE platform. J2EE platform have component based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications.

EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability and high performance. An EJB application can be deployed on any of the application server compliant with J2EE 1.3 standard specification.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EJB Component** | **Description** | **Stateless Session** | **Stateful Session** | **Entity** | **MDB** |
| Remote interface | The remote interface exposes business logic to remote clients—clients running in a separate application from the EJB. It defines the business methods a remote client can call. | Yes | Yes | Yes | No |
| Local interface | The local interface exposes business logic to local clients—those running in the same application as the EJB. It defines the business methods a local client can call.   |  |  | | --- | --- | | **Note:** | Not available for 1.1 EJBs. | | Yes | Yes | Yes | No |
| Local home interface | The local home interface, also referred to as an EJB factory or life-cycle interface, provides methods that local clients—those running in the same application as the EJB—can use to create, remove, and in the case of an entity bean, find instances of the bean.  The local home interface also has “home methods”—business logic that is not specific to a particular bean instance. | Yes | Yes | Yes | No |
| Remote home interface | The remote home interface, also referred to as an EJB factory, or life-cycle interface, provides methods that remote clients—those running in a separate application from the EJB—can use to create, remove, and find instances of the bean. | Yes | Yes | Yes | No |
| Bean class | The bean class implements business logic. | Yes | Yes | Yes | Yes |
| Primary key class | Only entity beans have a primary key class. The primary key class maps to one or more fields in a database—identifying the persistent data to which the entity bean corresponds. | No | No | Yes | No |

**T****he EJB Container**

An EJB container is a run-time container for beans that are deployed to an application server. The container is automatically created when the application server starts up, and serves as an interface between a bean and run-time services such as:

* + Life-cycle management
  + Code generation
  + Persistence management
  + Security
  + Transaction management
  + Locking and concurrency control

**EJB De****ployment Descriptors**

The structure of a bean and its run-time behavior are defined in one or more XML deployment descriptor files. Programmers create deployment descriptors during the EJB packaging process, and the descriptors become a part of the EJB deployment when the bean is compiled.

WebLogic Server EJBs have three deployment descriptors:

* + ejb-jar.xml—The standard J2EE deployment descriptor. All beans must be specified in an ejb-jar.xml. An ejb-jar.xml can specify multiple beans that will be deployed together.
  + weblogic-ejb-jar.xml—WebLogic Server-specific deployment descriptor that contains elements related to WebLogic Server features such as clustering, caching, and transactions. This file is required if your beans take advantage of WebLogic Server-specific features. Like ejb-jar.xml, weblogic-ejb-jar.xml can specify multiple beans that will be deployed together.
  + weblogic-cmp-jar.xml—WebLogic Server-specific deployment descriptor that contains elements related to container-managed persistence for entity beans. Entity beans that use container-managed persistence must be specified in a weblogic-cmp-jar.xml file.

**COBRA(Common Object Request Broker Architecture)**

## Common Object Request Broker Architecture (CORBA) is an architecture and specification for creating, distributing, and managing distributed program objects in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an "interface broker."

The essential concept in CORBA is the Object Request Broker (ORB). ORB support in a network of clients and servers on different computers means that a client program (which may itself be an [object](http://searchsoa.techtarget.com/definition/object)) can request services from a server program or object without having to understand where the server is in a distributed network or what the interface to the server program looks like. To make requests or return replies between the ORBs, programs use the General Inter-ORB Protocol (GIOP) and, for the Internet, its Internet Inter-ORB Protocol ([IIOP](http://searchcio-midmarket.techtarget.com/definition/IIOP)). IIOP maps GIOP requests and replies to the Internet's Transmission Control Protocol ([TCP](http://searchnetworking.techtarget.com/definition/TCP)) layer in each computer.

A notable hold-out from CORBA is Microsoft, which has its own distributed object architecture, the Distributed Component Object Model ([DCOM](http://whatis.techtarget.com/definition/DCOM-Distributed-Component-Object-Model)). However, CORBA and Microsoft have agreed on a gateway approach so that a client object developed with the Component Object Model will be able to communicate with a CORBA server (and vice versa).

The CORBA specification dictates there shall be an ORB through which an application would interact with other objects. This is how it is implemented in practice:

1. The application simply initializes the ORB, and accesses an internal *Object Adapter*,which maintains things like reference counting, object (and reference) instantiation policies, and object lifetime policies.
2. The Object Adapter is used to register instances of the*generated code classes*. Generated code classes are the result of compiling the user IDL code, which translates the high-level interface definition into an OS- and language-specific class base for use by the user application. This step is necessary in order to enforce CORBA semantics and provide a clean user process for interfacing with the CORBA infrastructure.

**Key features of this models.**

**COM**

1. COM objects can be used with all .NET languages through .NET [COM Interop](https://en.wikipedia.org/wiki/COM_Interop).
2. Defines a binary standard for component interoperability
3. Is programming-language-independent
4. Is provided on multiple platforms (Microsoft® Windows®, Windows 95, Windows NT�, Apple® Macintosh®, and many varieties of UNIX®)
5. Provides for robust evolution of component-based applications and systems
6. Is extensible by developers in a consistent manner
7. Uses a single programming model for components to communicate within the same process, and also across process and network boundaries
8. Allows for shared memory management between components
9. Provides rich error and status reporting
10. Allows dynamic loading and unloading of components

**EJB(Enterprise Java Beans)**

1. . An EJB application can be deployed on any of the application server compliant with J2EE 1.3 standard specification.
2. EJB components are server-side components written entirely in the Java programming language
3. EJB architecture is wire-protocol neutral--any protocol can be utilized like IIOP,JRMP, HTTP, DCOM,etc.
4. EJB components contain business logic only - no system-level programming & services, such as transactions, security, life-cycle, threading, persistence, etc. are automatically managed for the EJB component by the EJB server.
5. EJB architecture is inherently transactional, distributed, portable multi-tier, scalable and secure.
6. EJB components are fully portable across any EJB server and any OS.

**CORBA(Common Object Request Bracker Architecture)**

1. **Language independence**

CORBA was designed to free engineers from limitations of coupling their designs to a particular software language. Currently there are many languages supported by various CORBA providers, the most popular being Java and C++. There are also C++11, C-only, SmallTalk, Perl, Ada, Ruby, and Python implementations.

1. **OS-independence**

CORBA's design is meant to be OS-independent. CORBA is available in Java (OS-independent), as well as natively for Linux/Unix, Windows, Solaris, OS X, OpenVMS, HPUX, Android, LynxOS, VxWorks, ThreadX, INTEGRITY, and others.

1. **Freedom from technologies**

One of the main implicit benefits is that CORBA provides a neutral playing field for engineers to be able to normalize the interfaces between various new and legacy systems. When integrating C, C++, Object Pascal, Java, Fortran, Python, and any other language or OS into a single cohesive system design model, CORBA provides the means to level the field and allow disparate teams to develop systems and unit tests that can later be joined together into a whole system. This does not rule out the need for basic system engineering decisions, such as threading, timing, object lifetime, etc. These issues are part of any system regardless of technology. CORBA allows system elements to be normalized into a single cohesive system model  
**c) Data-typing**

CORBA provides flexible data typing, for example an "ANY" datatype. CORBA also enforces tightly coupled datatyping, reducing human errors. In a situation where Name-Value pairs are passed around, it is conceivable that a server provides a number where a string was expected. CORBA Interface Definition Language provides the mechanism to ensure that user-code conforms to method-names, return-, parameter-types, and exceptions.

1. **High tunability**

Many implementations (e.g. ORBexpress (Ada, C++, and Java implementation)[[1]](https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture" \l "cite_note-1) and OmniORB (open source C++ and Python implementation))[[2]](https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture#cite_note-2) have options for tuning the threading and connection management features. Not all ORB implementations provide the same features.

1. **Freedom from data-transfer details**

When handling low-level connection and threading, CORBA provides a high level of detail in error conditions. This is defined in the CORBA-defined standard exception set and the implementation-specific extended exception set. Through the exceptions, the application can determine if a call failed . Not receiving an exception means that the method call completed successfully. This is a very powerful design feature.

1. **Compression**

CORBA marshals its data in a binary form and supports compression. IONA, Remedy IT, and [Telefónica](https://en.wikipedia.org/wiki/Telefonica" \o "Telefonica) have worked on an extension to the CORBA standard that delivers compression. This extension is called ZIOP and this is now a formal OMG standard.