WIKIPEDIA

Common Object Request Broker Architecture

The **Common Object Request Broker Architecture** (**CORBA**) is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms. CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented. CORBA is an example of the distributed object paradigm.

Common Object Request Broker Architecture

Status	Published
Year started	1991
Latest version	3.3 October 2012
Organization	Object Management Group
Abbreviation	CORBA
Website	corba.org (http://corba.org/)

Contents

Overview

Versions history

Servants

Features

- Objects By Reference
- Data By Value
- Objects By Value (OBV)
- CORBA Component Model (CCM)
- Portable interceptors
- General InterORB Protocol (GIOP)
- VMCID (Vendor Minor Codeset ID)
- Corba Location (CorbaLoc)

Benefits

Problems and criticism

See also

References

Further reading

External links

Overview

CORBA enables communication between software written in different languages and running on different computers. Implementation details from specific operating systems, programming languages, and hardware platforms are all removed from the responsibility of developers who use CORBA. CORBA normalizes the method-call semantics between application objects residing either in the same address-space (application) or in remote address-spaces (same host, or remote host on a network). Version 1.0 was released in October 1991.

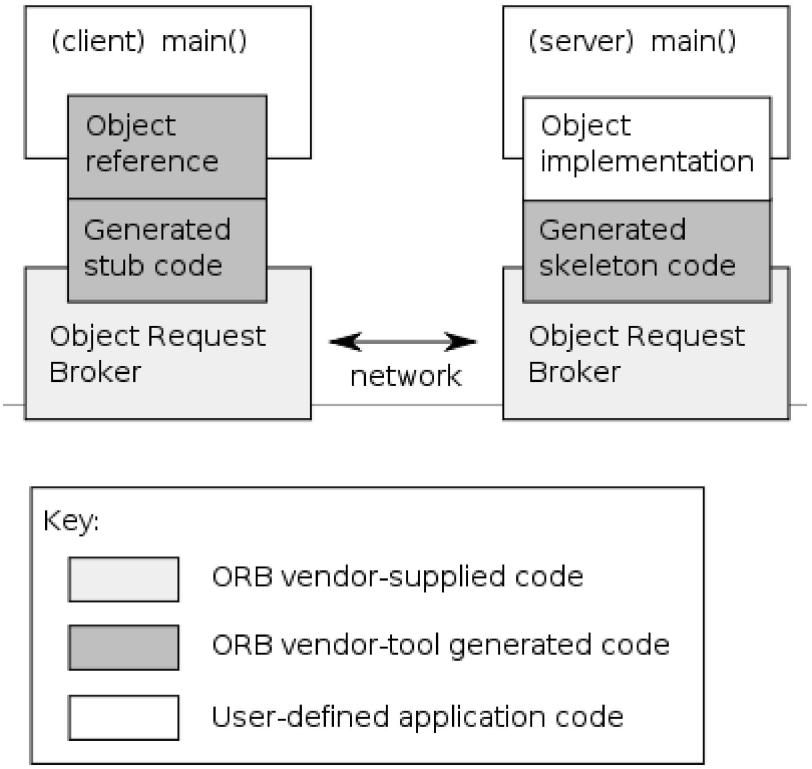
CORBA uses an interface definition language (IDL) to specify the interfaces that objects present to the outer world. CORBA then specifies a *mapping* from IDL to a specific implementation language like C++ or Java. Standard mappings exist for Ada, C, C++, C++11, COBOL, Java, Lisp, PL/I, Object Pascal, Python, Ruby and Smalltalk. Non-standard mappings exist for C#, Erlang, Perl, Tcl and Visual Basic implemented by object request brokers (ORBs) written for those languages.

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.

Some IDL mappings are more difficult to use than others. For example, due to the nature of Java, the IDL-Java mapping is rather straightforward and makes usage of CORBA very simple in a Java application. This is also true of the IDL to Python mapping. The C++ mapping requires the programmer to learn datatypes that predate the C++ Standard Template Library (STL). By contrast, the C++11 mapping is easier to use, but requires heavy use of the STL. Since the C language is not object-oriented, the IDL to C mapping requires a C programmer to manually emulate object-oriented features.

In order to build a system that uses or implements a CORBA-based distributed object interface, a developer must either obtain or write the IDL code that defines the object-oriented interface to the logic the system will use or implement. Typically, an ORB implementation includes a tool called an IDL compiler that translates the IDL interface into the target language for use in that part of the system. A traditional compiler then compiles the generated code to create the linkable-object files for use in the application. This diagram illustrates how the generated code is used within the CORBA infrastructure:



This figure illustrates the high-level paradigm for remote interprocess communications using CORBA. The CORBA specification further addresses data typing, exceptions, network protocols, communication timeouts, etc. For example: Normally the server side has the Portable Object Adapter (POA) that redirects calls either to the local servants or (to balance the load) to the other servers. The CORBA specification (and thus this figure) leaves various aspects of distributed system to the application to define including object lifetimes (although reference counting semantics are available to applications), redundancy/fail-over, memory management, dynamic load balancing, and application-oriented models such as the separation between display/data/control semantics (e.g. see Model–view–controller), etc.

In addition to providing users with a language and a platform-neutral remote procedure call (RPC) specification, CORBA defines commonly needed services such as transactions and security, events, time, and other domain-specific interface models.

Versions history

This table presents the history of CORBA standard versions.[1][2]

Version	Version Date	Highlights
1.0	October 1991	First version, C mapping
1.1	February 1992	Interoperability, C++ mapping
1.2	December 1993	-
2.0	August 1996	First major update of the standard, also dubbed **CORBA 2**
2.1	August 1997	-
2.2	February 1998	Java mapping
2.3	June 1999	-
2.4	August 2000	-
2.5	September 2001	-
2.6	December 2001	-
3.0	July 2002	Second major update of the standard, also dubbed **CORBA 3** CORBA Component Model (CCM)
3.0.1	November 2002	-
3.0.2	December 2002	-
3.0.3	March 2004	-
3.1	January 2008	-
3.1.1	August 2011	Adopted as 2012 edition of ISO/IEC 19500
3.2	November 2011	-
3.3	November 2012	Addition of ZIOP

Servants

A **servant** is the invocation target containing methods for handling the remote method invocations. In the newer CORBA versions, the remote object (on the server side) is split into the **object** *(that is exposed to remote invocations)* and **servant** *(to which the former part forwards the method calls)*. It can be one *servant* per remote *object*, or the same servant can support several (possibly all) objects, associated with the given Portable Object Adapter. The *servant* for each

object can be set or found "once and forever" (servant activation) or dynamically chosen each time the method on that object is invoked (servant location). Both servant locator and servant activator can forward the calls to another server. In total, this system provides a very powerful means to balance the load, distributing requests between several machines. In the object-oriented languages, both remote *object* and its *servant* are objects from the viewpoint of the object-oriented programming.

Incarnation is the act of associating a servant with a CORBA object so that it may service requests. Incarnation provides a concrete servant form for the virtual CORBA object. Activation and deactivation refer only to CORBA objects, while the terms incarnation and etherealization refer to servants. However, the lifetimes of objects and servants are independent. You always incarnate a servant before calling activate_object(), but the reverse is also possible, create_reference() activates an object without incarnating a servant, and servant incarnation is later done on demand with a Servant Manager.

The *Portable Object Adapter* (POA) is the CORBA object responsible for splitting the server side remote invocation handler into the remote *object* and its *servant*. The object is exposed for the remote invocations, while the servant contains the methods that are actually handling the requests. The servant for each object can be chosen either statically (once) or dynamically (for each remote invocation), in both cases allowing the call forwarding to another server.

On the server side, the POAs form a tree-like structure, where each POA is responsible for one or more objects being served. The branches of this tree can be independently activated/deactivated, have the different code for the servant location or activation and the different request handling policies.

Features

The following describes some of the most significant ways that CORBA can be used to facilitate communication among distributed objects.

Objects By Reference

This reference is either acquired through a stringified Uniform Resource Locator (URL), NameService lookup (similar to Domain Name System (DNS)), or passed-in as a method parameter during a call.

Object references are lightweight objects matching the interface of the real object (remote or local). Method calls on the reference result in subsequent calls to the ORB and blocking on the thread while waiting for a reply, success or failure. The parameters, return data (if any), and exception data are marshaled internally by the ORB according to the local language and OS mapping.

Data By Value

The CORBA Interface Definition Language provides the language- and OS-neutral inter-object communication definition. CORBA Objects are passed by reference, while data (integers, doubles, structs, enums, etc.) are passed by value. The combination of Objects-by-reference and data-by-value provides the means to enforce strong data typing while compiling clients and servers, yet preserve the flexibility inherent in the CORBA problem-space.

Objects By Value (OBV)

Apart from remote objects, the CORBA and RMI-IIOP define the concept of the OBV and Valuetypes. The code inside the methods of Valuetype objects is executed locally by default. If the OBV has been received from the remote side, the needed code must be either *a priori* known for both sides or dynamically downloaded from the sender. To make this possible, the record, defining OBV, contains the Code Base that is a space-separated list of URLs whence this code should be downloaded. The OBV can also have the remote methods.

CORBA Component Model (CCM)

CORBA Component Model (CCM) is an addition to the family of CORBA definitions.[3] It was introduced with CORBA 3 and it describes a standard application framework for CORBA components. Though not dependent on "language dependent Enterprise Java Beans (EJB)", it is a more general form of EJB, providing four component types instead of the two that EJB defines. It provides an abstraction of entities that can provide and accept services through well-defined named interfaces called *ports*.

The CCM has a component container, where software components can be deployed. The container offers a set of services that the components can use. These services include (but are not limited to) notification, authentication, persistence and transaction processing. These are the most-used services any distributed system requires, and, by moving the implementation of these services from the software components to the component container, the complexity of the components is dramatically reduced.

Portable interceptors

Portable interceptors are the "hooks", used by CORBA and RMI-IIOP to mediate the most important functions of the CORBA system. The CORBA standard defines the following types of interceptors:

1. IOR interceptors mediate the creation of the new references to the remote objects, presented by the current server.
2. Client interceptors usually mediate the remote method calls on the client (caller) side. If the object Servant exists on the same server where the method is invoked, they also mediate the local calls.
3. Server interceptors mediate the handling of the remote method calls on the server (handler) side.

The interceptors can attach the specific information to the messages being sent and IORs being created. This information can be later read by the corresponding interceptor on the remote side. Interceptors can also throw forwarding exceptions, redirecting request to another target.

General InterORB Protocol (GIOP)

The GIOP is an abstract protocol by which Object request brokers (ORBs) communicate. Standards associated with the protocol are maintained by the Object Management Group (OMG). The GIOP architecture provides several concrete protocols, including:

1. Internet InterORB Protocol (IIOP) – The Internet Inter-Orb Protocol is an implementation of the GIOP for use over the Internet, and provides a mapping between GIOP messages and the TCP/IP layer.
2. SSL InterORB Protocol (SSLIOP) – SSLIOP is IIOP over SSL, providing encryption and authentication.
3. HyperText InterORB Protocol (HTIOP) – HTIOP is IIOP over HTTP, providing transparent proxy bypassing.
4. Zipped IOP (ZIOP) – A zipped version of GIOP that reduces the bandwidth usage.

VMCID (Vendor Minor Codeset ID)

Each standard CORBA exception includes a minor code to designate the subcategory of the exception. Minor exception codes are of type unsigned long and consist of a 20-bit "Vendor Minor Codeset ID" (VMCID), which occupies the high order 20 bits, and the minor code proper which occupies the low order 12 bits.

Minor codes for the standard exceptions are prefaced by the VMCID assigned to OMG, defined as the unsigned long constant CORBA::OMGVMCID, which has the VMCID allocated to OMG occupying the high order 20 bits. The minor exception codes associated with the standard exceptions that are found in Table 3-13 on page 3-58 are or-ed with OMGVMCID to get the minor code value that is returned in the ex_body structure (see Section 3.17.1, "Standard Exception Definitions," on page 3-52 and Section 3.17.2, "Standard Minor Exception Codes," on page 3-58).

Within a vendor assigned space, the assignment of values to minor codes is left to the vendor. Vendors may request allocation of VMCIDs by sending email to tagrequest@omg.org. A list of currently assigned VMCIDs can be found on the OMG website at: http://www.omg.org/cgi-bin/doc?vendor-tags

The VMCID 0 and oxfffff are reserved for experimental use. The VMCID OMGVMCID (Section 3.17.1, "Standard Exception Definitions," on page 3-52) and 1 through 0xf are reserved for OMG use.

The Common Object Request Broker: Architecture and Specification (CORBA 2.3) (http://www.omg.org/cgi-bin/doc?formal/98-12-01)

Corba Location (CorbaLoc)

Corba Location (CorbaLoc) refers to a stringified object reference for a CORBA object that looks similar to a URL.

All CORBA products must support two OMG-defined URLs: "`corbaloc:`" and "`corbaname:`". The purpose of these is to provide a human readable and editable way to specify a location where an IOR can be obtained.

An example of corbaloc is shown below:

```
corbaloc::160.45.110.41:38693/StandardNS/NameServer-POA/_root
```

A CORBA product may optionally support the "`http:`", "`ftp:`" and "`file:`" formats. The semantics of these is that they provide details of how to download a stringified IOR (or, recursively, download another URL that will eventually provide a stringified IOR). Some ORBs do deliver additional formats which are proprietary for that ORB.

Benefits

CORBA's benefits include language- and OS-independence, freedom from technology-linked implementations, strong data-typing, high level of tunability, and freedom from the details of distributed data transfers.

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, just to mention a few.

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.

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.

For example, the design of a multitier architecture is made simple using Java Servlets in the web server and various CORBA servers containing the business logic and wrapping the database accesses. This allows the implementations of the business logic to change, while the interface changes would need to be handled as in any other technology. For example, a database wrapped by a server can have its database schema change for the sake of improved disk usage or performance (or even whole-scale database vendor change), without affecting the external interfaces. At the same time, C++ legacy code can talk to C/Fortran legacy code and Java database code, and can provide data to a web interface.

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.

High tunability

Many implementations (e.g. ORBexpress (Ada, C++, and Java implementation)[4] and OmniORB (open source C++ and Python implementation))[5] have options for tuning the threading and connection management features. Not all ORB implementations provide the same features.

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 for reasons such as "Small problem, so try again", "The server is dead" or "The reference does not make sense." The general rule is: Not receiving an exception means that the method call completed successfully. This is a very powerful design feature.

Compression

CORBA marshals its data in a binary form and supports compression. IONA, Remedy IT, and Telefónica 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.

Problems and criticism

While CORBA delivered much in the way code was written and software constructed, it has been the subject of criticism.[6]

Much of the criticism of CORBA stems from poor implementations of the standard and not deficiencies of the standard itself. Some of the failures of the standard itself were due to the process by which the CORBA specification was created and the compromises inherent in the politics and business of writing a common standard sourced by many competing implementors.

Initial implementation incompatibilities

The initial specifications of CORBA defined only the IDL, not the on-the-wire format. This meant that source-code compatibility was the best that was available for several years. With CORBA 2 and later this issue was resolved.

Location transparency

CORBA's notion of location transparency has been criticized; that is, that objects residing in the same address space and accessible with a simple function call are treated the same as objects residing elsewhere (different processes on the same machine, or different machines). This is a fundamental design flaw,[7] as it makes all object access as complex as the most complex case (i.e., remote network call with a wide class of failures that are not possible in local calls). It also hides the inescapable differences between the two classes, making it impossible for applications to select an appropriate use strategy (that is, a call with 1µs latency and guaranteed return will be used very differently from a call with 1s latency with possible transport failure, in which the delivery status is potentially unknown and might take 30s to time out).

Design and process deficiencies

The creation of the CORBA standard is also often cited for its process of design by committee. There was no process to arbitrate between conflicting proposals or to decide on the hierarchy of problems to tackle. Thus the standard was created by taking a union of the features in all proposals with no regard to their coherence.[8] This made the specification complex, expensive to implement entirely, and often ambiguous.

A design committee composed of a mixture of implementation vendors and customers created a diverse set of interests. This diversity made difficult a cohesive standard. Standards and interoperability increased competition and eased customers' movement between alternative implementations. This led to much political fighting within the committee and frequent releases of revisions of the CORBA standard that some ORB implementors ensured were difficult to use without proprietary extensions.[6] Less ethical CORBA vendors encouraged customer lock-in and achieved strong short-term results. Over time the ORB vendors that encourage portability took over market share.

Problems with implementations

Through its history, CORBA has been plagued by shortcomings in poor ORB implementations. Unfortunately many of the papers criticizing CORBA as a standard are simply criticisms of a particularly bad CORBA ORB implementation.

CORBA is a comprehensive standard with many features. Few implementations attempt to implement all of the specifications,[8] and initial implementations were incomplete or inadequate. As there were no requirements to provide a reference implementation, members were free to propose features which were never tested for usefulness or implementability. Implementations were further hindered by the general tendency of the standard to be verbose, and the common practice of compromising by adopting the sum of all submitted proposals, which often created APIs that were incoherent and difficult to use, even if the individual proposals were perfectly reasonable.

Robust implementations of CORBA have been very difficult to acquire in the past, but are now much easier to find. The SUN Java SDK comes with CORBA built-in. Some poorly designed implementations have been found to be complex, slow, incompatible and incomplete. Robust commercial versions began to appear but for significant cost. As good quality free implementations became available the bad commercial implementations died quickly.

Firewalls

CORBA (more precisely, GIOP) is not tied to any particular communications transport. A specialization of GIOP is the Internet Inter-ORB Protocol or IIOP. IIOP uses raw TCP/IP

connections in order to transmit data.

If the client is behind a very restrictive firewall or transparent proxy server environment that only allows HTTP connections to the outside through port 80, communication may be impossible, unless the proxy server in question allows the HTTP CONNECT method or SOCKS connections as well. At one time, it was difficult even to force implementations to use a single standard port — they tended to pick multiple random ports instead. As of today, current ORBs do have these deficiencies. Due to such difficulties, some users have made increasing use of web services instead of CORBA. These communicate using XML/SOAP via port 80, which is normally left open or filtered through a HTTP proxy inside the organization, for web browsing via HTTP. Recent CORBA implementations, though, support SSL and can be easily configured to work on a single port. Some ORBS, such as TAO, omniORB and JacORB also support bidirectional GIOP, which gives CORBA the advantage of being able to use callback communication rather than the polling approach characteristic of web service implementations. Also, most modern firewalls support GIOP & IIOP and are thus CORBA-friendly firewalls.

See also

Software engineering

- Component-based software engineering
- Distributed computing
- Portable object
- Service-oriented architecture (SOA)

Component-based software technologies

- Freedesktop.org D-Bus — current open cross-language cross-platform object model
- GNOME Bonobo — deprecated GNOME cross-language object model
- KDE DCOP — deprecated KDE interprocess and software componentry communication system
- KDE KParts — KDE component framework
- Component Object Model (COM) — Microsoft Windows-only cross-language object model
- DCOM (Distributed COM) — extension making COM able to work in networks
- Common Language Infrastructure — Current .NET cross-language cross-platform object model
- XPCOM (Cross Platform Component Object Model) — developed by Mozilla for applications based on it (e.g. Mozilla Application Suite, SeaMonkey 1.x)
- IBM System Object Model SOM and DSOM — component systems from IBM used in OS/2 and AIX
- Internet Communications Engine (ICE)
- Java remote method invocation (Java RMI)
- Java Platform, Enterprise Edition (Java EE)
- JavaBean
- OpenAIR
- Remote procedure call (RPC)
- Windows Communication Foundation (WCF)
- Software Communications Architecture (SCA) — components for embedded systems, cross-language, cross-transport, cross-platform

Language bindings

- Language binding
- Foreign function interface
- Calling convention
- Dynamic Invocation Interface

- Name mangling
- Application programming interface - API
- Application binary interface - ABI
- Comparison of application virtual machines
- SWIG opensource automatic interfaces bindings generator from many languages to many languages

References

1. "History of CORBA" (http://www.omg.org/gettingstarted/history_of_corba.htm). Object Management Group. Retrieved 12 March 2017.
2. "History of CORBA" (http://www.corba.org/history_of_corba.htm). Object Management Group. Retrieved 4 June 2017.
3. "The CORBA Component Model" (http://www.drdobbs.com/the-corba-component-model/184403888). Dr. Dobb's Journal. 1 September 2004. Retrieved 13 March 2017.
4. "ORBexpress : Real-time CORBA ORB" (http://www.ois.com/Products/communications-middleware.html).
5. "omniORB : Free CORBA ORB" (http://omniorb.sourceforge.net). *sourceforge.net*. Retrieved 9 January 2014.
6. Chappel, David (May 1998). "Trouble with CORBA" (https://web.archive.org/web/20121203074323/http://www.davidchappell.com/articles/article_Trouble_CORBA.html). www.davidchappel.com. Archived from the original (http://www.davidchappell.com/articles/article_Trouble_CORBA.html) on 3 December 2012. Retrieved 15 March 2010.
7. Waldo, Jim; Geoff Wyant; Ann Wollrath; Sam Kendall (November 1994). "A Note on Distributed Computing" (http://www.cc.gatech.edu/classes/AY2010/cs4210_fall/papers/smli_tr-94-29.pdf) (PDF). *Sun Microsystem Laboratories*. Retrieved 4 November 2013.
8. Henning, Michi (30 June 2006). "The Rise and Fall of CORBA" (http://queue.acm.org/detail.cfm?id=1142044). *ACM Queue*. Association for Computing Machinery. **4** (5). Retrieved 15 March 2010.

Further reading

- "CORBA" (https://www.omg.org/spec/CORBA/). *Current*. Specification. OMG.
- Orfali, Robert. *The Essential Client/Server Survival Guide* (https://archive.org/details/essentialclients00orfa). John Wiley & Sons. ISBN 0-471-15325-7.
- Orfali, Robert; Harkey, Dan; Edwards, Jeri. *The Essential Distributed Objects Survival Guide*. John Wiley & Sons. ISBN 0-471-12993-3.
- Orfali, Robert; Harkey, Dan. *Client/Server Programming with JAVA and CORBA*. John Wiley & Sons. ISBN 0-471-24578-X.
- Slama, Dirk; Garbis, Jason; Russell, Perry. *Enterprise CORBA*. Prentice Hall. ISBN 0-13-083963-9.
- Henning, Michi; Vinoski, Steve. *Advanced CORBA Programming with C++*. Addison-Wesley. ISBN 0-201-37927-9.
- Korthaus, Axel; Schader, Martin; Aleksy, Markus. *Implementing Distributed Systems with Java and CORBA* (https://web.archive.org/web/20051031102447/http://www.wifo.uni-mannheim.de/CORBA/). Springer. ISBN 3-540-24173-6. Archived from the original (http://www.wifo.uni-mannheim.de/CORBA/) on 31 October 2005. Retrieved 23 June 2005.
- Bolton, Fintan. *Pure Corba*. Sams Publishing. ISBN 0-672-31812-1.
- Siegel, Jon. *CORBA 3 - Fundamentals and Programming*. John Wiley & Sons. ISBN 0-471-29518-3.
- Zahavi, Ron. *Enterprise Application Integration with CORBA: Component and Web-Based Solutions*. John Wiley & Sons. ISBN 0-471-32720-4.
- Hartman, Bret; Beznosov, hartman; Vinoski, Steve; Flinn, Donald. *Enterprise Security with EJB and CORBA*. John Wiley & Sons. ISBN 0-471-40131-5.
- Mowbray, Thomas J.; Zahavi, Ron. *The Essential Corba: System Integration Using Distributed Objects*. John Wiley & Sons. ISBN 0-471-10611-9.
- Rosen, Michael; Curtis, David. *Integrating CORBA and COM Applications*. John Wiley & Sons. ISBN 0-471-19827-7.
- Brose, Gerald; Vogel, Andreas; Duddy, Keith. *Java Programming with CORBA*. John Wiley & Sons. ISBN 0-471-37681-7.
- Schettino, John; Hohman, Robin S.; O'Hara, Liz. *CORBA For Dummies*. Hungry Minds. ISBN 0-7645-0308-1.
- Rosenberger, Jeremy L. *Teach Yourself CORBA in 14 Days*. Sams Publishing. ISBN 0-672-31208-5.

- Siegel, Jon. *Quick CORBA 3*. John Wiley & Sons. ISBN 0-471-38935-8.
- Mowbray, Thomas J.; Malveau, Raphael C. *CORBA Design Patterns*. John Wiley & Sons. ISBN 0-471-15882-8.
- Orfali, Robert; Harkey, Dan; Edwards, Jeri. *Instant CORBA*. John Wiley & Sons. ISBN 0-471-18333-4.
- Harmon, Paul; Morrissey, William (1996). *The Object Technology Casebook*. John Wiley & Sons. ISBN 0-471-14717-6.

External links

- Official OMG CORBA Components page (http://www.omg.org/spec/CCM/)
- Unofficial CORBA Component Model page (http://ditec.um.es/~dsevilla/ccm)
- Comparing IDL to C++ with IDL to C++11 (http://taox11.remedy.nl/documents/presentations.html)
- Corba: Gone But (Hopefully) Not Forgotten (http://queue.acm.org/detail.cfm?id=1388786)
- OMG XMI Specification (http://www.omg.org/spec/CORBA/)