A TAXONOMY FOR COMPARING DISTRIBUTED OBJECT TECHNOLOGIES

A. Bracho, A. Matteo, Ch. Metzner
Centro de Ingeniería de Software Y Sistemas ISYS,
Facultad de Ciencias, Universidad Central de Venezuela,
Apdo. 48093, Caracas 1041-A, Venezuela
e-mail: amatteo/cmetzner@isys.ciens.ucv.ve

Abstract

This works goal is the definition of a set of criteria to be used as a reference for evaluation and comparison of existing distributed object technologies. In particular the defined criteria are applied to the evaluation of: CORBA, DCOM and RMI but they can also be applied to other distributed object technologies or to compare implementations of a particular one. The results of the our evaluation are discussed and presented.

Key words: distributed systems, distributed objects, CORBA, COM, DCOM, RMI.

1. Introduction

Nowadays, due to the existing technologies for distributed objects, the possibility to develop distributed applications with components located on any machine in a network is a reality. It is a well-known fact that there are three leading technologies sharing the market: the *Object Management Group* (OMG), with more than 800 members, promotes an architecture specification for the administration of distributed objects (*Object Management Architecture*). On the other hand, Microsoft has incorporated mechanisms for distributed objects (*Distributed Component Object Model)* in the Windows NT/95 operating system. Finally, JavaSoft from Sun Microsystems extended the Java language with a complete architecture for distributed objects: *Remote Method Invocation*.

Considering the diversity of technologies, criteria are needed to provide a framework for technology selection. [Alho97], [Chung97], [Orfali95b] and [Orfali97] compare some technologies, but they do not define general criteria that allow a comparison. This fact motivated the present work, were a set of criteria is defined and their application to the evaluation and comparison of CORBA/Orbix, DCOM/Microsoft and RMI/SunSoft is presented.

Besides this introduction and the conclusions, this work is structured in the following sections: section 2, 3 and 4 describe briefly the distributed object technologies: CORBA, DCOM and RMI. In section 5 general criteria for evaluation and comparison are defined and in section 6 CORBA, DCOM and RMI are evaluated and compared.

2. Common Object Request Broker Architecture (CORBA)

CORBA is a standard specification providing an architecture for the development of distributed object-oriented applications [Otte96]. It is part of the *Object Management Architecture (OMA)* defined by the OMG and it specifies the *Object Request Broker* (ORB) component of OMA.

CORBA was defined in 1990. It's 1.1 version was released in December 1991 and describes how to develop a CORBA implementation; it defined also the *Interface Definition Language* (IDL) and the *Application Programming Interface* (API) that allow the client / server interaction in a specific ORB implementation. CORBA 2.0 and the *Internet Inter-Orb Protocol* (IIOP*)* transport protocol was released in December 1994 and it specifies how ORBs of different vendors should interact. Until this point, CORBA only defined the interaction between distributed objects of one

and the same vendor. In 1998 the OMG adopted the CORBA 2.2 specification and the CORBA 2.3 spec is now under development.

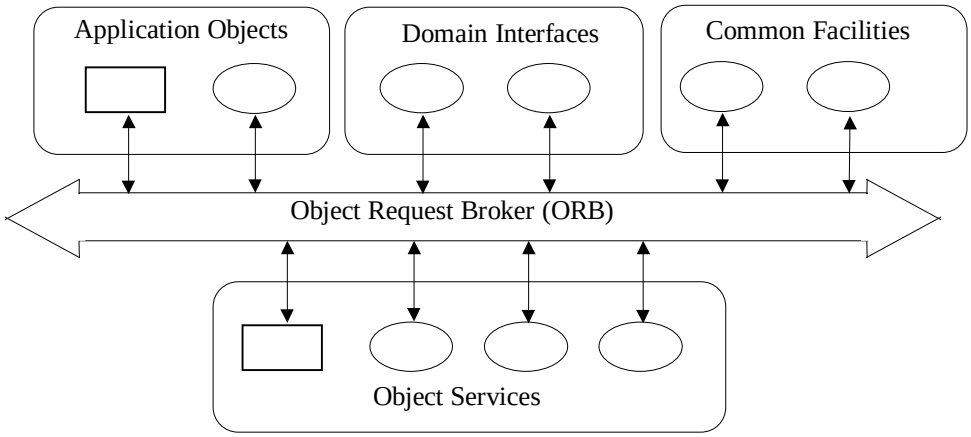


Figure 1. Components of the OMA.

The **Object Management Architecture (OMA)** has five components, as shown in Figure 1: the ORB *Object Request Broker* (ORB), *Object Services*, *Common Facilities*, *Domain Interfaces* and *Application Objects.*

The ORB is the middleware that establishes the client / server relationships between objects independently of the platform and specific techniques. A client object can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for locating an object that can implement the request, pass the parameters, invoke the corresponding method and return the results. For the client, the object location, the programming language, operating system or any aspect not part of the object's interface is transparent.

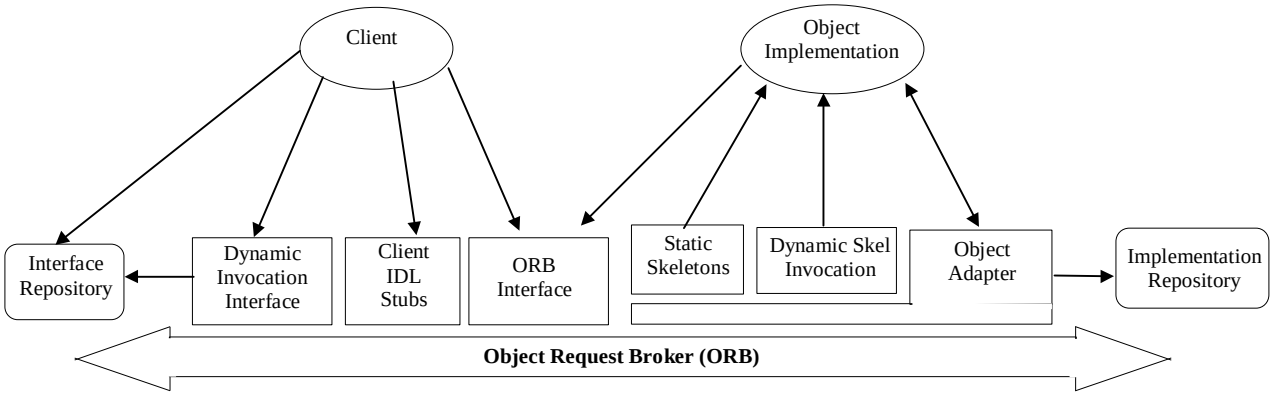


Figure 2. Structure of a CORBA 2.0 ORB.

Figure 2 shows the client and server sides of a CORBA ORB. The components of an ORB are [Orfali 95b]:

a) On the client side:

- ***Interface Repository (IR).*** Allows to obtain and modify the descriptions of all registered component interfaces, the methods they support, and the parameters they require.
- ***Dynamic Invocation Interface (DII).*** The dynamic invocation API allows to generate and invoke object requests at run time.
- ***Client IDL Stubs***. Provide the static interfaces to the services of a remote server object.
- ***ORB Interface***. Consists of a few APIs to local services that may be of interest to an application.

b) On the server side:

- ***Static Skeletons***. Provide static interfaces to each service exported by the server.
- ***Dynamic Skel Invocation*** (DSI). Provides a run-time binding mechanism for servers that need to handle incoming method calls for components that do not have IDL-based compiled skeletons.
- ***Object Adapter (OA)***. Accepts requests for service on behalf of the server's objects. It provides the run-time environment for instantiating server objects, passing requests to them, and assigning object IDs.
- ***Implementation Repository***. Provides a run-time repository of information about the classes a server supports, the objects that are instantiated and their IDs.
- ***ORB Interface***. Consists of a few APIs to local services that are identical to those provided on the client side.

3. Distributed Component Object Model (DCOM)

DCOM is a communication protocol for software components on a network [DCOM96]. It is the distributed version of the *Component Object Model* (COM), a Microsoft binary standard describing the binary format of executable files and defining object interaction [Grime97].
The first version (1996) of DCOM is part of Microsoft's Windows NT 4.0 and it is expected that DCOM will become an integral part of future Windows versions. In 1997 Microsoft announced the development of COM+ which integrates COM, DCOM and the *Microsoft Transaction Server* (MTS). In COM+ there is no difference between local and remote objects.
DCOM consists of a set of extensions and layers build on the *Distributed Computing Environment* (DCE) from *Open Group*. It is based on the specifications of the *Open Software Foundation* DCE-RPC. Due to its relation to *Remote Procedure Call* (RPC) specification, it is also called *Object RPC* (ORPC) [DCOM96].
In Figure 3 the DCOM architecture is shown. The *COM Run-Time* provides remote object methods to clients and components. The DCE RPC and the *Security Provider* are used to generate a standard network package according to the DCOM specifications.

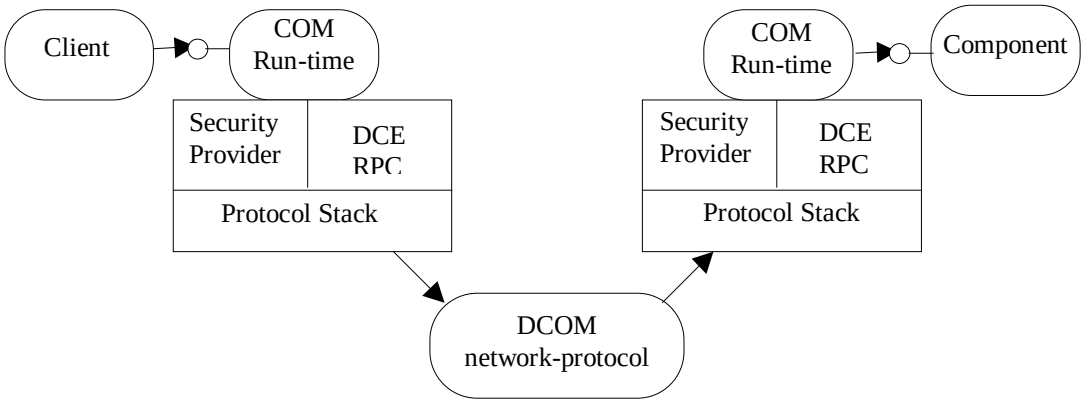


Figure 3. DCOM Architecture.

4. Remote Method Invocation (RMI)

RMI defines and supports a distributed object model for the Java language hiding the ORB from the programmer and providing an API for the development of distributed applications. *SunSoft* developed the API for applications using RMI and it comes with the *Java Development Kit* (JDK). RMI allows the creation of remote objects using Java interfaces. A remote object in Java is an object whose methods are invoked from an other Java Virtual Machine (JVM). It was designed for Java / Java applications, i.e. Java is used for implementing objects on the client and on the server [RMI97a], [RMI98], [Sun 98].

Three abstraction levels define the RMI architecture (Figure 4): the stub/skeleton, the remote reference and the transport level. *Stub/skeleton* has two components: the *stub* (on the client side) and the *skeleton* (on the server's side). The *stub* and the *skeleton* are automatically generated from the server class using the compiler *rmic*. The s*tub* organizes and serializes the object's parameters providing the client with an interface and a behavior that are consistent with the remote object. The *skeleton* communicates directly with the object implementation and the remote reference level of the server. The *remote reference level* determines the invocation protocol, which is independent of the stubs and skeletons. Every remote object uses an invocation protocol as: point to point, duplicate object groups, a specific duplication and reconnection strategy. For example, the remote reference level determines when the server is a simple or duplicate object and in the later case communication with the different copies is required.

The *transport level* is responsible of sending data between a client and a server, obtaining the correct address of an object when it is invoked, administer and monitor the connection between client and server. If a connection is interrupted or if some error occurs when transferring data he notifies the remote reference level using exceptions.

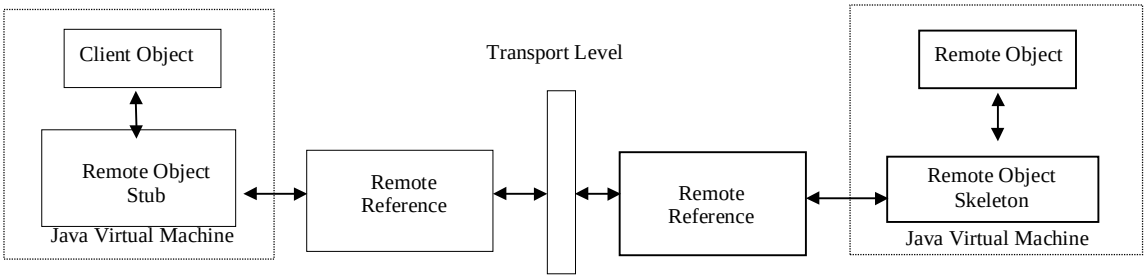


Figure 4. RMI Architecture

5. Criteria

In this section a general set of criteria is defined that allow a comparison between different distributed object technologies. The definition is based on the technical specification of: CORBA, DCOM and RMI [OOTM95], [Chau97], [Albert98].

(a) ***Language Interoperability.*** Specifies if the technology allows communication between objects implemented in different programming languages.

(b) ***Platform Interoperability***. Specifies if the technology allows communication between objects implemented in different operating systems and heterogeneous platforms.

(c) ***Interface Definition Language***. Describes the properties of the interface definition language provided by the technology.

(c.1) ***Mechanism for construction of interfaces***. Specifies the mechanisms used by the IDL to define an interface.

(c.2) ***Relationship between an interface and the implementation of a class.*** This relationship can be one-to-one or many-to-one. The last one means that a class is implemented from multiple interface definitions.

(d) ***Protocols***. Specifies the underlying communication protocols used by the technology.

(e) ***Garbage Collector.*** In a distributed system, a garbage collector on the server must be able to eliminate automatically remote objects that are not referenced by clients. This criterion identifies if the technology uses a garbage collector and its properties.

(f) ***Parameter transmission.*** The parameter transmission mechanism used by a technology is described. Particularly these can be:

(f.1) ***By value.*** An object can be transmitted by value. In this case a copy of the transmitted object will be on the server.

(f.2) ***By reference***. An object can be transmitted by reference. In this case a reference to the object on the server is transmitted.

(f.3) ***Primitive data types.*** Identifies the mechanisms used for transmitting primitive data types (integer, real, floating, etc.) as parameters.

(g) ***Remote method access***. Identifies if the invocation is static or dynamic, and if there is a difference between a local or remote method invocation.

(g,1) ***Static invocation.*** In this case the name of the server object, the name of the invoked method and its parameters have to be known by the client program at compile time.

(g.2) ***Dynamic invocation.*** Allows the localization of an object, the invoked method and its parameters at execution time.

(h) ***Interface repository.*** Indicates if the technology has an interface repository.

(i) ***Portability***. Independence of the platform at source code and / or executable code level is indicated.

(j) ***Commercial criteria.*** Subdivides into commercial value subcriteria that must be periodically updated.

(j.1) ***Operating systems***. Indicates the operating systems for which an implementation of the technology is available.

(j.2) ***Programming languages.*** The different languages for which a technology specifies mappings. The mappings explain how to use IDL constructions in a particular programming language.

(k) ***Performance.*** Response time for method activation is evaluated using as parameters the following types and data structures.

Parameters	Description
No parameters	Method activation with no parameter transmission
Byte	8 bit
Boolean	boolean
Char	character
Short	16 bit signed integer
Integer	32 bit signed integer
Long	64 bit signed integer
Float	Floating point simple precision
Double	Floating point double precision
All the primitive types	Parameters for the activated method are byte, boolean, char, short, integer, long float and double
String	255 characters
Integer array	Array of 32 bit signed integers
Struct	Struct with the following data types: byte, boolean, char, short, integer, long, float and double

Table 1. Parameters for Method Activation

6. Evaluation and Comparisson of CORBA, DCOM and RMI

Using the previously defined criteria a comparison of the technologies is discussed next and the results are presented in Table 2. Note that CORBA is a standard specification implemented by various vendors. In this work the CORBA 2.0 implementation considered is Orbix from *IONA Technologies* and it is compared to DCOM 95/NT from *Microsoft* and RMI from *SunSoft*.

(A) *Interoperability at language level.*

CORBA/Orbix and DCOM are technologies that are independent of a programming language. CORBA is a standard specification that can be implemented in any programming language. DCOM supports the most known languages: C, C++, Delphi, Visual Basic and Java (J++), but other languages can also be used with DCOM with calls to the binary standard COM. RMI only allows applications where client and server are implemented in Java, but through the API *Java Native Interface* (JNI) native methods written in C or C++ can be invoked. If a RMI client wants to communicate with a remote object implemented in C or C++, it is necessary to define an object adapter between them (Figure 5).

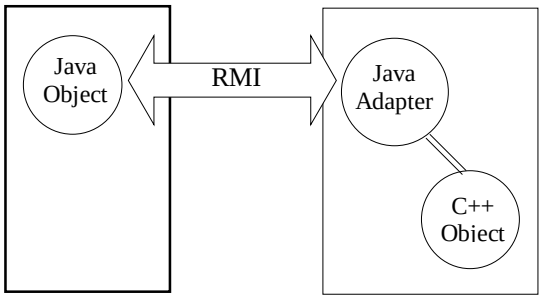


Figure 5. Native methods in Java.

RMI is integrated in Java; they represent a “programming technology” [Cur97]. On the other side RMI, DCOM and CORBA/Orbix represent an “integration technology not a programming de technology” [Cur97]; they are designed as an integrating base for different programming technologies.

(B) Platform Interoperability

CORBA/Orbix, DCOM and RMI allow communication between objects implemented on different platforms and heterogeneous operating systems. For DCOM there are implementations for some operating systems (see (J.1)).

(C) Interface Definition Language

CORBA/Orbix and DCOM separate interfaces and their implementations through an IDL. OMG´s and Microsoft's (MIDL) IDLs are incompatible and based on different object models. In CORBA/Orbix all interfaces are specified using the IDL. In DCOM the IDL is not strictly necessary since a standard for binary interface is provided. DCOM offers also an *Object Description Language* (ODL), a subset of the IDL, that generates metadata for the interface that an object supports while the IDL generates *stubs*. In RMI the functionality of the IDL is provided through the interfaces of Java.

(C.1) Interface construction mechanism

For defining new interfaces the IDL of CORBA/Orbix and RMI/Java [Van97] provide simple and multiple inheritance; since DCOM does not provide in its IDL an inheritance mechanism [Chau97] to define an interface in terms of other interfaces, aggregation and interface packaging is used [Orfali95].

(C.2) Relationship between interface and implementation class

In CORBA/Orbix a one-to-one relationship exists between an interface definition and its implementation, while in DCOM and RMI a class can be implemented from more than one interface definition.

(D) Protocols

CORBA/Orbix uses the *Internet Inter-ORB Protocol* (IIOP) for communication assuring thereby interoperability between ORBs.

DCOM uses the DCE RPC standard under TCP/IP for communicating COM objects on remote machines.

Initially RMI/Java only used the *Java Remote Method Protocol* (JRMP) under TCP/IP to communicate remote objects. RMI/Java also uses the IIOP [Sun98] for interoperability between a CORBA implementation and RMI/Java.

(E) Garbage Collector

CORBA/Orbix does not provide a garbage collector.DCOM provides a garbage collector [DCOM96] that uses a reference counter for objects referenced by one or more clients. RMI Java automatically provides the garbage collector.

(F) Parameter transmission

(F.1) By value

CORBA/Orbix does not provide by value parameter transmission. A proposal for adopting by value transmission is included in the CORBA 2.3 specification [OMG98], [OMG96].

DCOM allows passing objects by value [Ray97] and in RMI local objects are transmitted by value. When a local object is transmitted to a virtual machine, a new object is created in it and by defect, only non static attributes of the transmitted object are copied [RMI97a].

(F.2) By reference

CORBA/Orbix and DCOM provide by reference transmission. In RMI a reference to the corresponding *stub* is passed when a remote object is transmitted [RMI97a]. In this case the client interacts with a local object, the *stub*, that represents the remote object.

(F.3) Primitive data type transmission

Primitive data type transmission under CORBA/Orbix and DCOM can be by value, by reference or by result; RMI has by value transmission for primitive data types. Transmission by result means that after executing a method the final values are copied onto the actual parameters.

(G) Remote method access

At programming level there is no distinction between a local and remote object.

(G.1) Static invocation

CORBA/Orbix, DCOM and RMI allow definition of static invocations at compile time.

(G.2) Dynamic invocation

CORBA/Orbix and DCOM allow method binding at execution time while RMI does not allow dynamic invocation.

(H) Interface repository

DCOM has the *Type Library* as an interface repository. The equivalent to DCOM´s *Type Library* in CORBA/Orbix is the *Interface Repository* (IR) containing all interface and metadata definitions. RMI has no interface repository [RMI98]; all the objects are defined in Java and the class file generated by the language describes their types, methods and attributes.

(I) Portabilidad

CORBA/Orbix and DCOM programs are portable but hey have to be recompiled for every different platform. As the Java compiler generates bytecodes that are portable, Java/RMI programs are portable.

(J) Commercial criteria

(J.1) Operating systems

CORBA/Orbix implementations are available for more than 20 operating systems: UNIX (Solaris, HP/UX, AIX, IRIX, OSF/1, SunOS, Solaris x86, UnixWare, SCO, Sinix and Ultrix) Windows For Workgroups 3.11, Windows 95, Windows NT, OS/2, VMS, pSOS, QNX, VxWorks and Macinstosh.

DCOM is only available for Windows 95/NT but there are beta versions for other platforms. In 1995 Microsoft signed an agreement with *Software AG* to assure portability of COM/DCOM to non-Windows operating systems particularly to Linux (Caldera), DG/UX (Data General), HP-UX (*Hewlett-Packard)*, AIX (IBM), Solaris (Sun), OS/400, Open VMS (Digital) MVS (IBM). Due to the Microsoft – Apple alliance in 1998, versions of COM/DCOM for Macintosh environmments are being implemented.

Java and RMI are available for Unix/Solaris and Windows 95/NT.

(J.2) Programming languages

CORBA/Orbix specifies mappings for the most popular programming languages: C++, Java, Cobol and Ada. The OMG also provides source code generation for various languages using IDL files or directly from an interface repository. DCOM provides mappings for C++ and Java in its IDL. But it is possible to build applications without using the IDL, using COM´s binary standard, which describes executable files format independently of the programming language used.

RMI is designed for building applications only in Java but the JNI (*Java Native Interface*) mechanism allows calls to native code written in other languages.

TECHNOLOGY / CRITERIA		CORBA/ORBIX	DCOM 95/NT	RMI/JDK 1.1.4
Language Interoperability		Yes	Yes	No
Platform Interoperability		Yes	Yes	Yes
IDL		IDL from OMG	IDL/ODL Microsoft	Java Interfaces
	Interface Construction Mechanism	Multiple Inheritance	Aggregation Packaging	Multiple Inheritance
	Interface – Implementation Relationsship	One-to-one	Many-to-one	Many-to-one
Protocols		IIOP	ORPC	JRMP/IIOP
Garbage Collector		No	Yes	Yes
Parameter Transmission	***objects by value***	No	Yes	Local objects only
	objects by reference	Yes	Yes	Only remote objects (Stub reference)
	Primitive data types	By Value, Reference, Result	By Value, Reference, Result	By Value
Remote Method Access	***Static invocation***	Yes	Yes	No
	Dynamic invocation	Yes	Yes	No
Interface Repository		*Interface Repository* (IR)	*Type Library*	No
Portability		Source Code	Source code	Source / object code
Commercial Criteria	***Operating systems***	Windows 95/NT, OS/2, VMS, pSOS, QNX, VxWorks, Macinstosh	Windows 95/NT	Unix/Solaris, Windows 95/NT
	Programming languages	C++, Cobol, Ada, Java.	C++, Java,Visual Basic	Java

Table 2. Comparisson

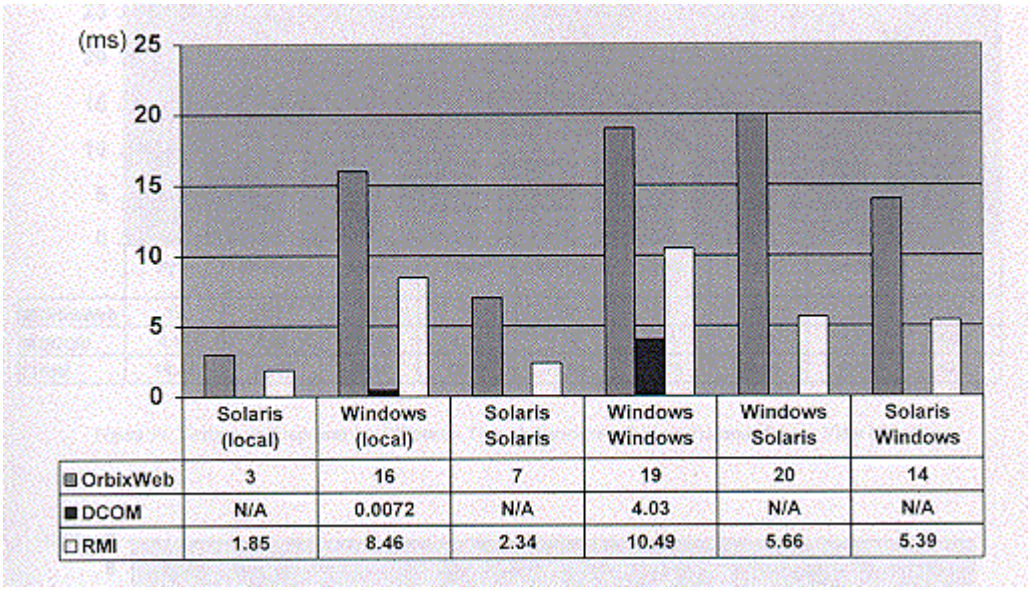
(*K) Performance*

The evaluation program has one client and one server and was executed on three different platforms:

- On a *Ultra2*, 200 Mhz, 128 Mb RAM, with SunSolaris 5.1 for the OrbixWeb and RMI implementations.
- On a Pentium Intel 75 Mhz. 16 Mb RAM, Windows 95 for the OrbixWeb, RMI implementations and a DCOM client.
- On a Pentium Intel 133 Mhz, 64 Mb RAM, Windows NT 4.0 as a DCOM server

Because of technical restrictions, for DCOM the client and server programs were executed on different operating systems. The program was run on a *TCP/IP* LAN during low traffic hours.
The JDK 1.1.4 and OrbixWeb 3.0 *Standard Edition* from IONA was used as a CORBA implementation. Visual J++ 1.1 and the Win32 SDK version 2.01 for Java were used for version 1.1 of DCOM 95.

For each method activation by a client the response time was calculated using the parameters described in Table 1. The methods are services from the server.
Response time for a method activation without parameters was less for DCOM then for CORBA/Orbix or RMI. RMI activations were better than OrbixWeb on all platforms and

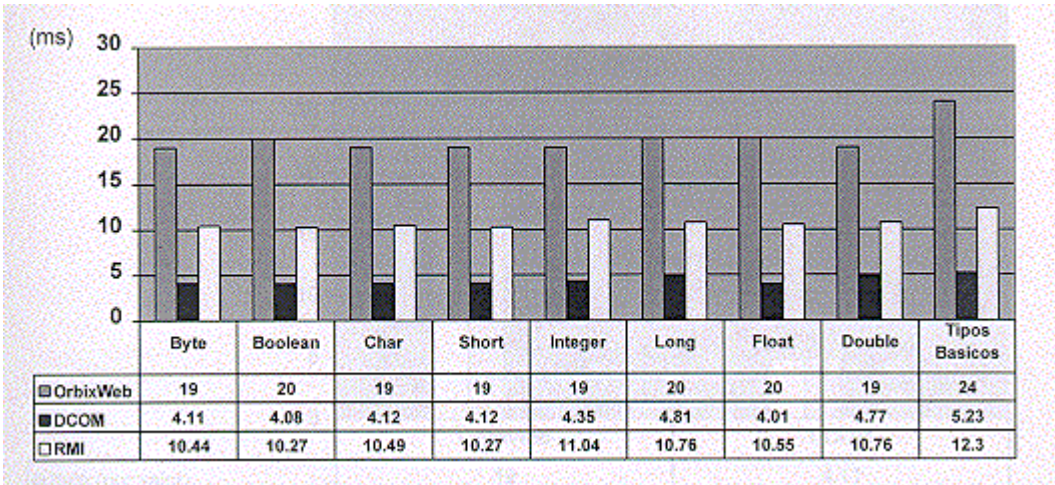


	Solaris (local)	Windows (local)	Solaris Solaris	Windows Windows	Windows Solaris	Solaris Windows
OrbixWeb	3	16	7	19	20	14
DCOM	N/A	0.0072	N/A	4.03	N/A	N/A
RMI	1.85	8.46	2.34	10.49	5.66	5.39

OrbixWeb had the worst response times. Response times for remote activations on Solaris-Solaris were better then on Windows-Windows. These results are shown in Figure 6.

Figure 6. Response time for activations

Figure 7 and 8 show the results for by value parameter transmission for Windows and Solaris respectively. DCOM had the best times followed by RMI and OrbixWeb under Windows. Under Solaris RMI response time was better than OrbixWeb.



	Byte	Boolean	Char	Short	Integer	Long	Float	Double	Tipos Basicos
OrbixWeb	19	20	19	19	19	20	20	19	24
DCOM	4.11	4.08	4.12	4.12	4.35	4.81	4.01	4.77	5.23
RMI	10.44	10.27	10.49	10.27	11.04	10.76	10.55	10.76	12.3

Figure 7. Response time for by value parameter transmission (Windows)

OrbixWeb and DCOM response times for by reference parameter transmission is worse than by result because Java has no by reference transmission of local objects and the OrbixWeb and DCOM implementations simulate it.

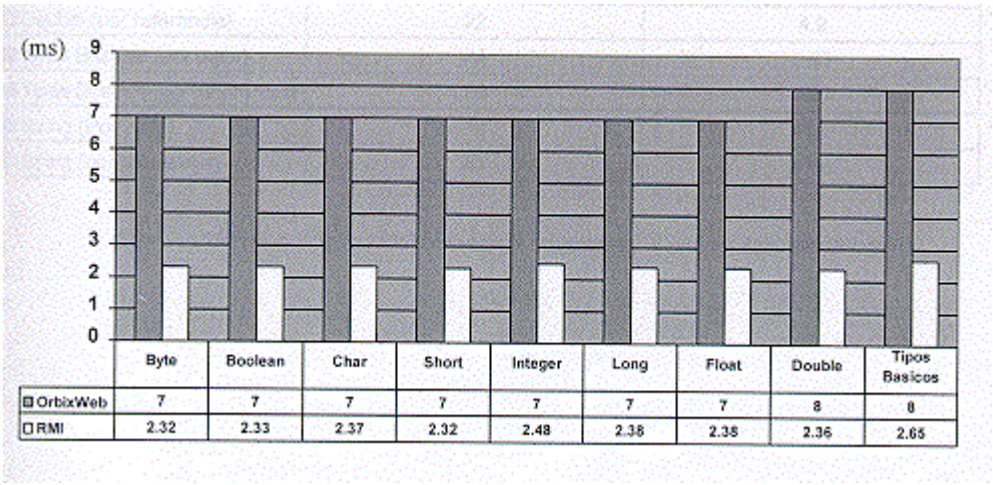


Figure 8. Reponse time for by value parameter transmisson (Solaris)

For OrbixWeb on Solaris response times for parameter transmission by value of primitive data types and by result were similar. For integer arrays by reference parameter transmission in OrbixWeb and DCOM is better than by value. RMI arrays are transmitted only by reference. The complete results are detailed in [BM 98].

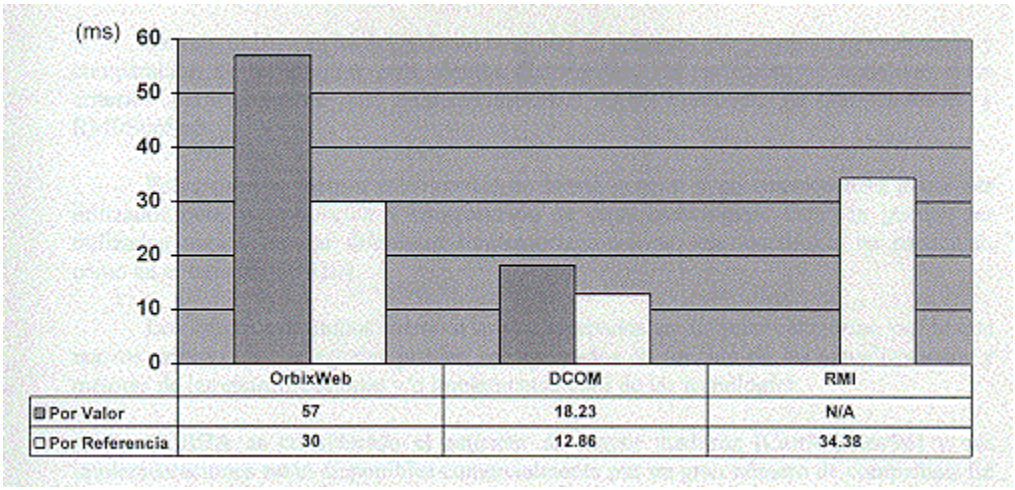
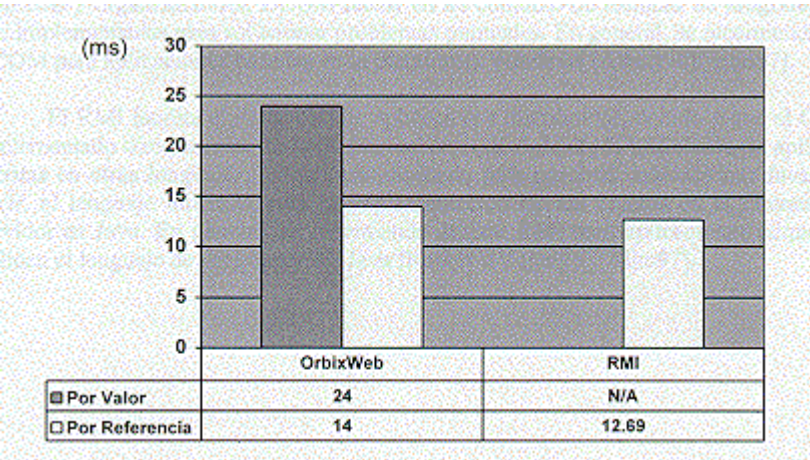


Figure 9. Response time for integer array parameter transmission (Windows)

Figure 10. Response time for integer array parameter transmission (Solaris)



7. Conclusions and Future Work

The defined criteria are general and can be used for evaluation and comparison of other distributed object technologies. They also may be applied to compare different implementations of a particular technology, for example CORBA. It is not an exhaustive list and it is possible to extend them, incorporate dynamic and performance criteria and / or classify them.This is part of our ongoing research.
It can be suggested that for the development of distributed applications with reuse of components written in different languages, CORBA is nowadays the first choice. For new applications RMI is well suited and DCOM is basically oriented to Windows NT platforms.

8. References

[Albert98] Albertson, Tom *Best practices in distributed object application development: RMI, CORBA and DCOM*. Tech Focus. Feb 23, 1998. http://www.developer.com/news/techfocus

[Alho97] Alho, Kari *Comparision of CORBA, DCOM and RMI*. Workshop on Emerging Tecnologies in Distributed Systems. Helsinki University of Technology. January,1998. http://wwwseg.cs.hut.fi/~ta/CORBA/Orbix-comparision/sld0001.htm

[BM 98] Bracho, Alexander; Matteo, Alfredo; *Definición de Criterios y Comparación de Tecnologías para Objetos Distribuidos.* Trabajo de Grado de Maestría en Ciencias de la Computación. Universidad Central de Venezuela, Caracas, Venezuela, Octubre 1998.

[Cur97] Curtis, David *Java, RMI and CORBA*. White Papper. OMG. 1997
.

[Chau97] Chauvet, Jean-Marie *CORBA, ActiveX y Java Beans.* Ediciones Gestión 200. 1997
.

[Chung97] Chung P.E. *DCOM and CORBA Side by Side, Step and Step, and Layer by Layer.* C++ Report Magazine. September, 1997
http://www.research.microsoft.com/os/ywang/papers/HTML/DCOMnCORBA/S.html

[DCOM96] *Distributed Component Object Model Protocol. DCOM/1.0.* Network Working Group. Nov, 1996.

[Grime97] Grimes, Richard *Professional DCOM programming.* Wrox Press. 1997.

[OMG96] *Objects By Value*. RFP. OMG Article No. orbos/96-06-14. http://www.omg.org/

[OMG98] *Objects By Value*. Joint Revised Submission. OMG Article No. *orbos/98-01-18.* http://www.omg.org/

[OOTM95] *A Distributed Object Premier.* OOTM Lab Publication. May, 95.
http://www.thomtech.com/~ootm/content/papers/DistObjPrimer.htm

[Orfali95] Orfalli, Robert; Harkey, Dan *Cliente/Servidor con Objetos Distribuidos.* Byte. Abril, 1995.

[Orfali95b] Orfalli, Robert; Harkey Dan; Edwards Jeri *Distributed Objects. Survival Guide.* John Wiley & Sons. 1995.

[**Orfali97]** Orfalli, Robert; Harkey, Dan *Client/Server Programming with JAVA and CORBA.*John Wiley & Sons. 1997.

[Orfali98] Orfalli Robert; Harkey Dan; Edwards, Jeri *Cliente/Servidor. Guía de supervivencia.* McGraw-Hill. Segunda Edición.1998.

[Otte96] Otte, Randy *Understanding CORBA/ORBIX.* Prentice Hall. 1996.

[Ray97] Raymond, Jayson *Java and Distributed Object Systems. Corbis Corp*.
http://www.seajug.org/html/DisObSys/TOC.htm

[RMI97a] *Java Remote Method Invocation Specification.* Revision 1.4. JDK 1.1 FCS, February 10, 1997.

[RMI98] *Frequently Asked Questions. RMI and Object Serialization*.
http://java.sun.com/products/jdk/rmi/faq.html

[Sun98] RMI and IIOP in Java. FAQ. SunSoft. March 1998.
http://java.sun.com/pr/1997/june/statement970626-01.faq.html

[Van97] Vanhelsuwé, Laurence *La biblia de Java.* Anaya Multimedia. 1997.