Compare Architecture of CORBA, RMI and DCOM

Faculty of Information Technology
Monash University

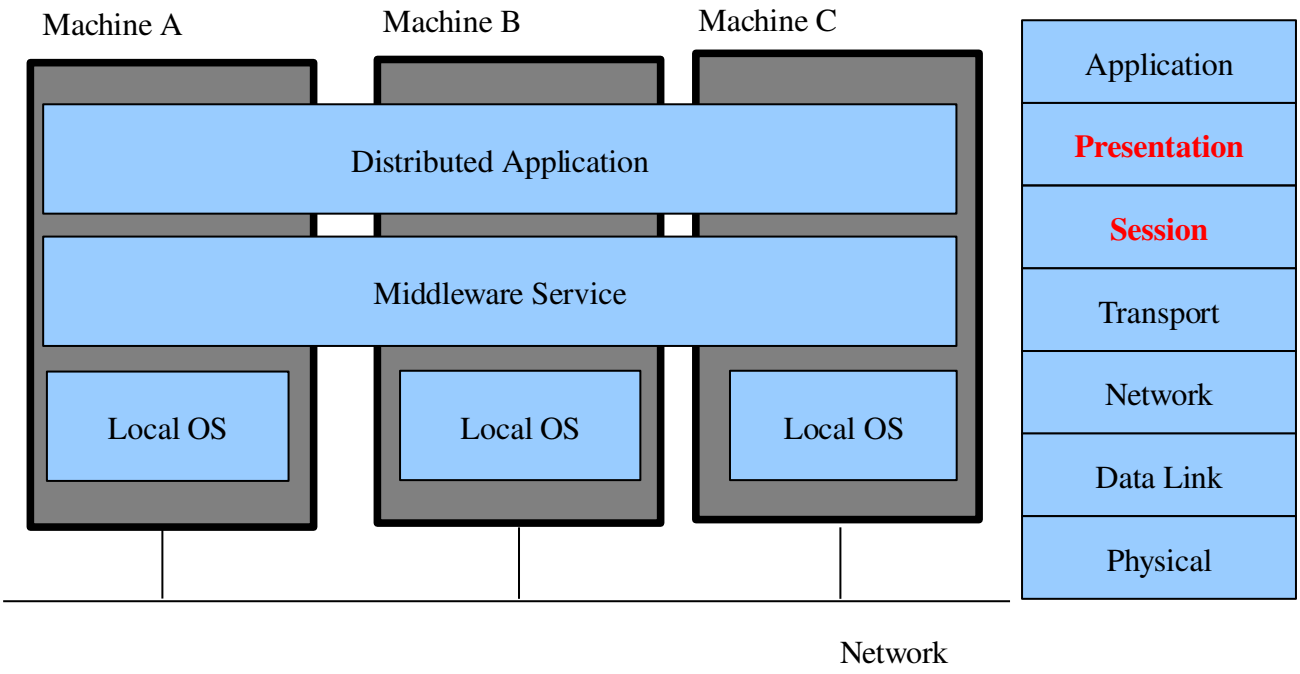
Student Name: Zihong Chen
Student ID: 21311242
Email: czihong@gmail.com

Abstract

In the current environment of IT industry, various hardware, platform, developing language and application are running on thousands of workstations and servers. It raises an issue that how to communicate and interoperate the diverse distributed application in the distributed system. The middleware technology is an approach to solve the heterogeneous problem of distributed system. The purposes of this paper are to overview the current middleware technology, and compare these technologies. The paper is divided into five sections. The first section, introduction, which talks what is distributed system and the role of middleware in the distributed system. Section two, as a bird’ eyes view of the industry standard of Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA). In the next two sections, it brief overview the Sun Microsystems’s Remote Method Invocation (RMI) technology, and Microsoft's Distributed Common Object Model (DCOM). After that, this paper compares the different between CORBA, RMI and DCOM.

Introduction

It is better to clarify what is distributed system and middleware before we go deeper. Various definition of distributed system is given around the world, most of them are correct. It is depended on from which aspect to look at the distributed system. To compare with the parallel system, Vijay claim that distributed system is multi processors that connect via network (Vijay, 2004, p.1). To compare with centralized system, Andrew gives a definition of distributed system (Andrew, 2002, p.2):“A distributed system is a collection of independent computers that appears to its users as a single coherent system.” Andrew also describes seven principles of distributed system: communication, processes, naming, synchronization, consistency and replication, fault tolerance and security. In addition, Andrew classifies four paradigms, distributed object-based systems, distributed file system, distributed document-based systems and distributed coordination-based (Andrew, 2002). In this paper, we prefer Andrew's definition, because we focus on the distributed object-based systems, which talk more about middleware. As the issue of how to communicate and interoperate the distributed application via the network, middleware plays an import role to communicate the distributed application. The figure 1 gives an overview of the relationship of distributed application, middleware, platform and network. We can say that middleware enable the communication of distributed application and object. From the view of ISO/OSI reference model, middleware can be trusted as a layer. The middleware layer bridges the gasp between the network operation system and application layer. The middleware is at presentation and session layer. Figure 1 A distributed system organized as middleware.



Wolfgang category middleware into three types: transaction-oriented middleware, message-oriented middleware and remote procedure calls. In the paper, we are not going to compare the different among these three types, for more detail that can find on the book (Wolfgang, 1999, p. 61-86).

Remote Procedure Calls (RPC) is a mechanism that invented by Sun Microsystems to perform the network communication. RPC is a part of Sun's Open Network Computing (ONC) platform. The idea of object-oriented middleware comes from the RPC. OMG firstly introduced the object-oriented middleware system as Common Object Request Broker Architecture (CORBA). After that, Microsoft support the distributed object capability via Component Object Model(COM), and Sun publish the Remote Method Invocation (RMI) in Java. The mechanisms of RPC and object oriented middleware are similar, but there are still some significant differences between them, the next section will explain it.

CORBA

CORBA is developed by OMG. The CORBA specification 1.0 was published in the beginning of the 1990s. CORBA provide the architecture and infrastructure that allow the application communicate with each other over

the network, whatever the platform or developing language. So far, the version of CORBA is 3.0, the specifications of CORBA can be downloaded from OMG official webpage (http://www.omg.org/cgi-bin/doc?formal/2004-03-01). OMG is the first company describes the object-oriented middleware technology, and CORBA becomes the standard in the industry. Figure 2 is the overview of OMG Object Management Architecture. The Common Object Services (CORBA services) include Naming, Persistence, Life Cycle, Concurrency, Event etc. Each service can be looked as interface. The detail of each service can be found on this link: http://www.omg.org/technology/documents/formal/corbaservices.htm .

The facilities in CORBA are a group of CORBA service. Vertical Facilities is a group of high level service. The purpose of the vertical facilities is to group the application domain, such as banking, telecom, manufacturing etc. Horizontal Facilities are a group of service which independent of application domain. In addition, horizontal facilities including user interface, for instance, information management, system management etc. Horizontal facilities are generally end-user-oriented.

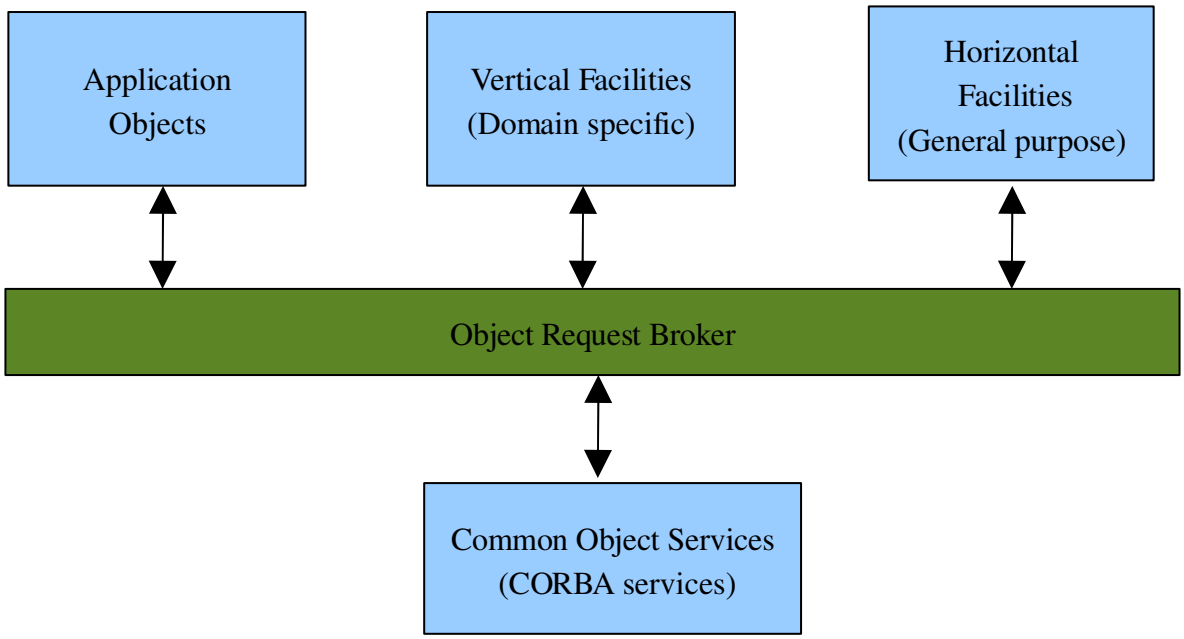


Figure 2 the overview of OMG Object Management Architecture

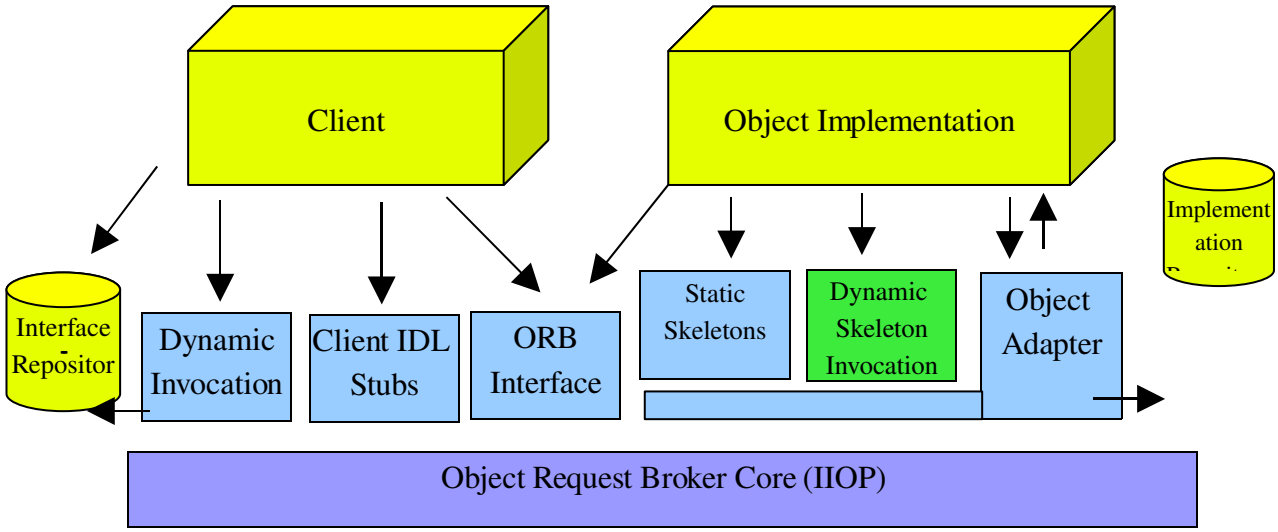


Figure 3 overview the CORBA

Internet Inter-ORB Protocol (IIOP) is a standard protocol specified by CORBA. The IIOP protocol is over TCP/IP, the client/server pass the request and respond via the IIOP. Figure 3 is the overview of CORBA.

Object Request Broker (ORB) bridges the grasp between client and server. It is a middleware that handle the request and respond of object over the network. In other word, ORB looks like an object bus. Over the ORB, a client object can communicate the object over the server side. Robert and Dan explain one benefit of using ORB compare with RPC (Robert & Dan, 1998, p.10). In RPC, a client object calls the function of object which the data is separate with in the server. In contract to RPC, a client calls separate object in ORB. Each object many have their own private data, the invocation in ORB calls separate object's function, and handle with the object's own data, do not interrupt other. This is because of the power of polymorphism. Figure 4 explain the benefit of ORB.

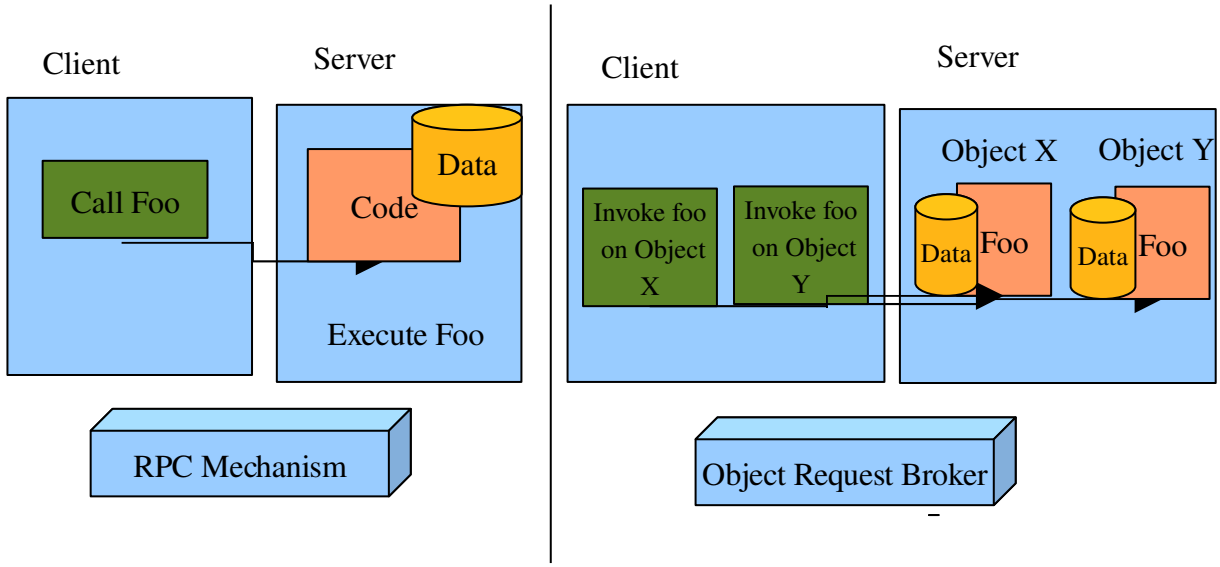


Figure 4 explain the benefit of ORB

CORBA IDL: IDL stands for Interface Definition Language, in CORBA, the object and services are defined in IDL. The main goal of IDL complier is to map IDL specifications to the existing programming language. For example, The OMG IDL define the CORBA object, the syntax and semantics of OMG IDL can find in the CORBA/IIOP specific (the specific document can downland on the officinal web page, http://www.omg.org/cgi-bin/doc?formal/2004-03-01 . Idlj is an IDL to java compiler, which map CORBA object into java. Table 1 lists the map from IDL to Java.

IDL Type	**Java Type**
module	package
boolean	boolean
char, wchar	char
octet	byte
string, wstring	java.lang.String
short, unsigned short	short
long, unsigned long	int
long long, unsigned	long

IDL Type	**Java Type**
long long	
float	float
double	double
fixed	java.math.BigDecimal
enum, struct, union	class
sequence, array	array
interface (non-abstract)	signature interface and an operations interface, helper class, holder class
interface (abstract)	signature interface, helper class, holder class
constant (not within an interface)	public interface
constant (within an interface)	fields in the Java signature interface for non-abstract, or the sole Java interface for abstract
exception	class
Any	org.omg.CORBA.Any
type declarations nested within interfaces	"scoped" package
typedef	helper classes
pseudo objects	pseudo interface
readonly attribute	accessor method

IDL Type	Java Type
readwrite attribute	accessor and modifer methods
operation	method

Table 1 map CORBA IDL to Java

Interface Repository and Implementation Repository: In the client side, interface repository contains the IDL information. In other word, Interface Repository is similar to a run time distributed database that store the registered component interfaces. Once the IDL defines the attributes, methods and parameters of object, the object is registered in the interface repository. So the client can invoke the remote method via the interface repository. In the server side, implementation repository is similar to interface repository. It contains the information to locate and activate an object once the server receives a request from client. Implementation repository contains information that allows the ORB to locate and activate implementations of objects.

Client IDL Stubs / Static Skeletons: the stub and skeletons are generated by the IDL compiler. The client IDL stubs define the client method to call the service on the server. The static skeletons provide an interface which respond to the request from client, and execute the service on the server side. Image the scene that the client stub marshal the request, and send the request via the ORB. Once the skeletons receive the request from the client, and then unmarshal it, and execute it.

Dynamic Invocation Interface / Dynamic Skeleton Interface: Dynamic Invocation allows client to construct an invocation request in run time, such as what object to implement, what method to invoke and what is the parameters. These invokes which are constructed by client, do not go though the client stub (which is static). The DII contains a generic invoke operation, send the client’s request to server. In the server side, dynamic skeleton interface

provide the mechanism to hand the request which is not IDL-based compiled stubs which means the request is not predefine.

ORB Interface: ORB interface provide some simple APIs to local service. Such as convert an object reference to a string. This capability is simple and powerful to communicate object references via ORB.

Object Adapters: Object Adapter is the core of CORBA. Object Adapter is a component that provides the capability to make object implement itself available to communicate via an ORB. The repository of object adapter is instantiating a server object, send request to object, and give the object a reference. There are two type of object adapter, one is Basic Object Adapter (BOA), and another is Portable Object Adapter (POA).

Implement

In this paper, it is not going to implement a CORBA application. To help understand the CORBA architecture, we brief describe the main steps to create a CORBA application with Java.

Figure 5 demonstrate the main steps to create a CORBA application (Robert & Dan, 1998, p.67-82).

1. Define your server interfaces using the interface definition language (IDL).
2. Bind the interface definition to the Interface Repository.
3. Run the IDL file through a language precompiled.
4. Add the servant implementation code.
5. Compile the code.
6. Register the run time objects with the implementation repository.
7. Instantiate the object on the server.
8. Implement the client code
9. Compile the client code.

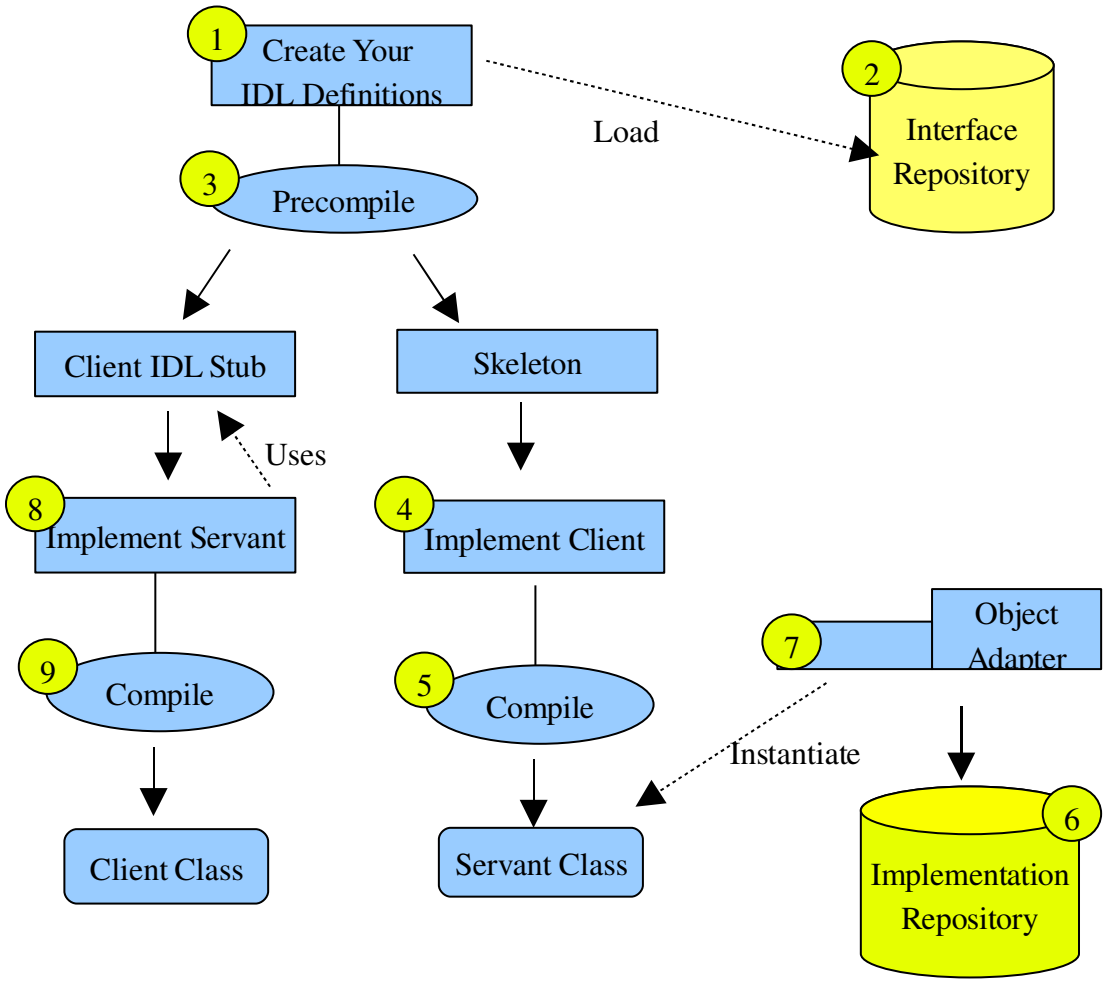


Figure 5 demonstrate the main steps to create a CORBA application

RMI

RMI is a java mechanism that enables the programmer to create a distributed java to java application. Figure 6 the overview of RMI architecture (Troy, 1998, p. 19-87).

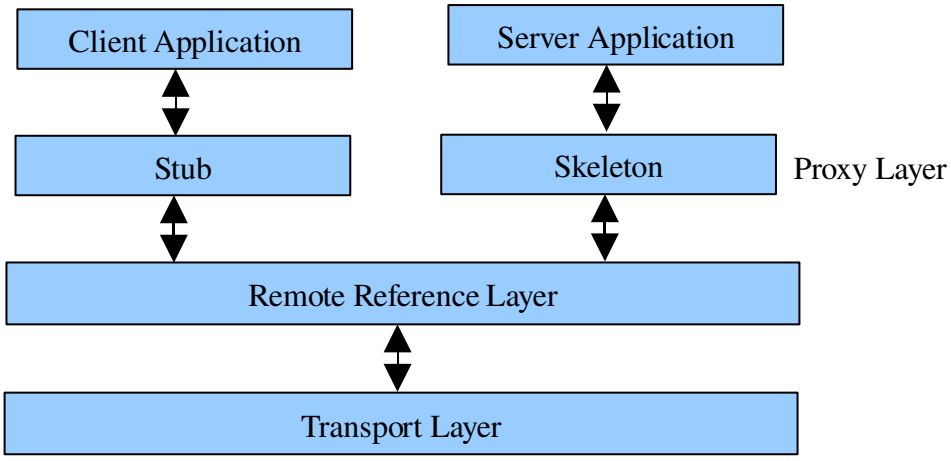


Figure 6 the overview of RMI architecture

Application Layer: It is a layer that server application is invoked by the remote client. The application need to extend the **Remote** interface. Java.rmi.Remote is an interface which does not contain any method. The Remote interface indicates weather an object is remotely accessible or not. In addition, the applications also need to register themselves with the Java naming service. The object will obtain a reference which is given by the naming service. Since the application is registered, the client can expose it via the reference.

In UNIX, the RMI registry can be launched by the following command: **rmiregistry &**. "&" indicate the rmiregistry service is running at the background. The default port of rmiregistry is 1099, to change the port number, type the following command: **rmiregistry 2099 &**. Mostly, the port number can be closed between 1024 and 16000, because the number which below 1024 is reserve for system service, such as ftp, telnet, http.

Figure 7 the reference of an object is formatted likes below:

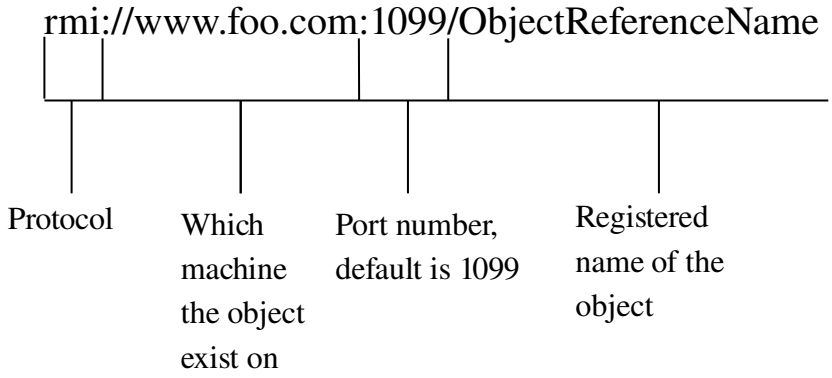


Figure 7 the reference of an object

Proxy Layer: The proxy layer contains stub in the client side and skeleton in the server side. The three distributed object oriented technology (CORBA, RMI and DCOM) exist the proxy layer, but Microsoft named the proxy layer with different names, for example DCOM call the stub in the client side as proxy. The more difference among these three technologies will explain in the following section shortly. Stub and skeleton contain the classed which are generated by RMI stud compiler (RMIC). The stub and skeleton are directly communicate with the remote reference layer. The proxy layer provides the

interface to the remote object. The responsibility of proxy layer contains: to initial a call to remote object, marshal the request data, unmarshal the data, and dispatch the object. In RMI, the stub and skeleton communicate each other via the marshal stream. The marshal stream is transported in the remote reference layer. In RMI, a marshal stream is an object, which contains the parameters, exceptions, error, etc. As we mentioned before, the stub can skeleton are generated by RMIC. The following commands show how to generate the skeleton: **$rmic -d /home/MyClasses rmibook.timeserver.TimeServerImpl**. Once the command is executed, a TimeServerImpl_Skel.class will generate in your current directory.

Transport Layer: the goal of transport layer is to make a connection from machine to machine via the TIC/IP protocol.

Remote Reference Layer: the remote reference layer is an abstract layer between proxy layer and transport layer. It is responsible to handle the marshal stream between proxy layer and transport layer. In other word, we can say the remote reference layer is a stream oriented layer.

DCOM

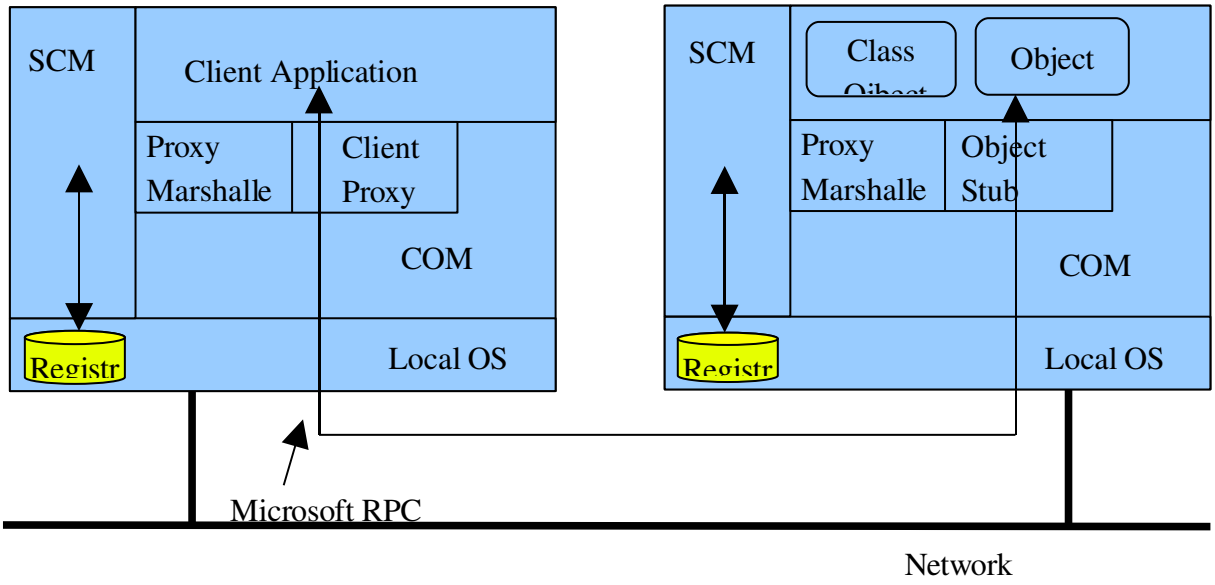


Figure 8 architecture of DCOM

DCOM is Microsoft's technology to adopt the remote object model. Figure 8 it the architecture of DCOM. It comes from COM. COM stand for Common Object Model. It could be a dynamically linkable library (dll), or executable

program. The gold of COM is to enable dynamically invoke and operate with each other.

As figure 8 display, the architecture of DCOM is similar to CORBA and RMI, such as the proxy layer in RMI, or client stub and serve skeleton in CORBA. In DCOM, the stub in client is named client proxy, and the server skeleton is named object stub. The client proxy and server skeleton is generated by Microsoft Interface Language (MIDL). The purpose of MIDL is the same as CORBA IDL. The MIDL generated some class file, and store in type library (which named interface repository in CORBA). A DCOM object is an implementation of an interface which is binary interface. A binary interface is generated by MIDL. In contract to CORBA, binary interface has an advantage. In CORBA, other programming languages need to follow the standard of CORBA, and map them to IDL specifications, such as Idlj which is IDL to java compiler. In DCOM, the binary interface is programming language independent, so they do not need to do what CORBA does (Andrew, 2002, p. 527). Figure 9 explains the benefit of DCOM.

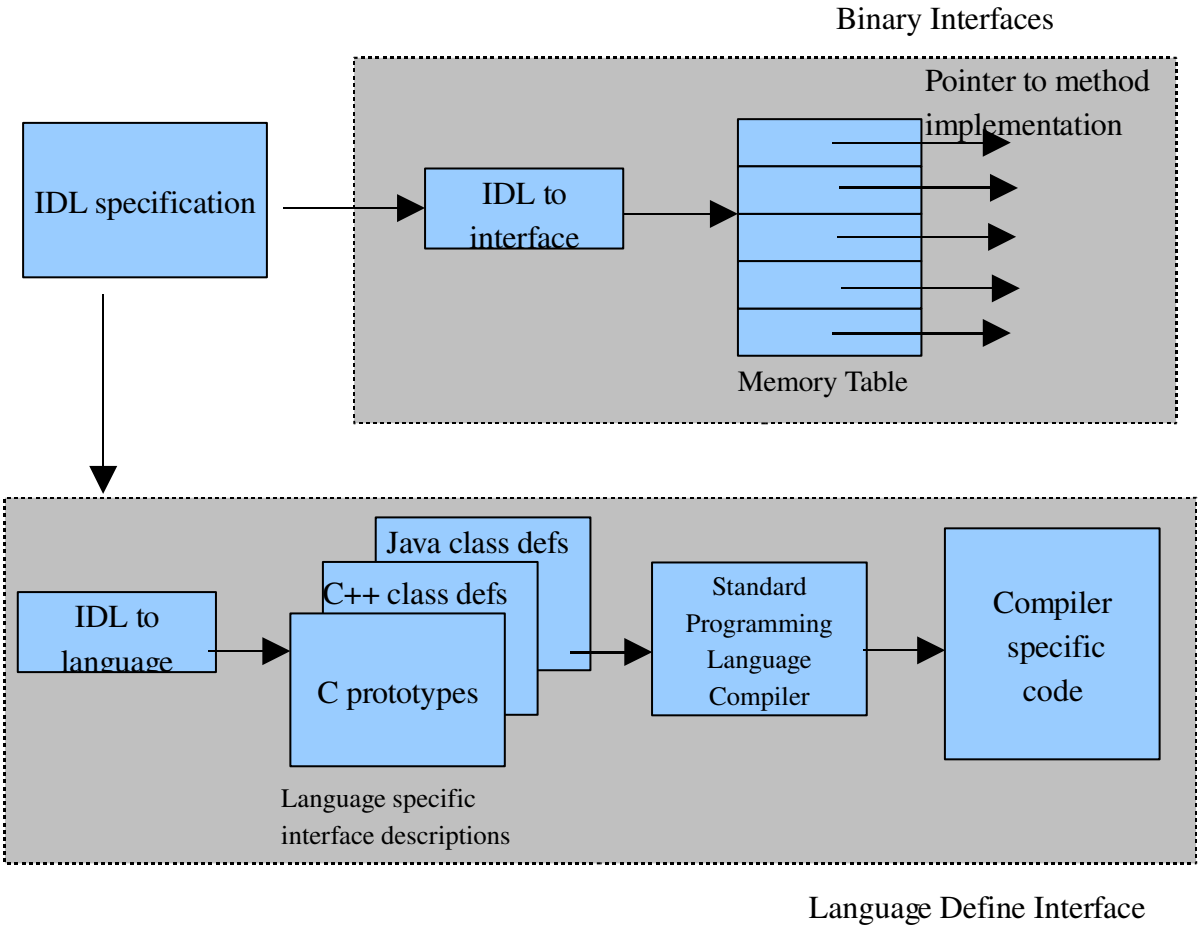


Figure 9 explain the benefit of DCOM

To create a DCOM object, the class need to implement the IClassFactory. The IClassFactory is defined in unknwn.idl file. The bellowing code explains how IClassFactory defined (Frank, 1997, p.5-20):

```
interface IClassFactory : Iunknown
{
        HRESULT CreateInstance([in, unique] Iunknown * pUnkOuter
                [in] REFIID riid,
                [out, iid_is(riid)] void ** ppvObject);
        HRESULT LockServer([in] BOOL flock;
}
```

In Microsoft architecture, there are two type of interface, one is standard interface, such as IUnknown and IClassFactory, and the other is customer interface which is defined by the programmer. The IClassFactory contain the CreatedInstance method, which create a class object when the method is invoked. The DCOM object has an Interface Identifier (IID). In Microsoft DCOM, the DCOM object is identified by the Class Identifier which is a special part of Globally Unique Identifiers (GUID). The GUID has 128-bit, the structure is showed below:

```
typederf struct GUID
{
        DWORD Data1;    //32-bits
        WORD Data2;     //16-bits
        WORD Data3;     //16-bits
        BYTE Data4[8];  //64-bits
}
```

One benefit of using GUID to identify the object instead of human readable name is to avoid the name conflict. The property of two object has the same GUID is nearly zero in theory. In the other hand, human readable name may be the same. When someone uses your DCOM object, the developer may be also defines the same object name. In systems such as Distributed COM, interfaces are specified at the lower level in the form of tables, there so called

binary interfaces, they are nature independent of any programming language.

Once a DCOM object is created and given a CLSID, this object will registered in the windows platform registry. The gold of windows registry in DCOM technology is mapping the CLSID into the object name. The object will be looked up via the registry. As we mentioned, the architecture of DCOM is similar to CORBA in some way, we can map DCOM type library into CORBA interface repository. On the other hand, we can map DCOM Service Control Manager (SCM) into CORBA implantation repository. The gold of SCM is to response the object request and activate object in the server side. The server receives a request from the client, and then looks up the registry via the CLSID, start a process which associated with the CLSID.

Compare CORBA, RMI and DCOM

To develop a distributed system, the performance of the system is one of the important aspects need to be considered. Roland compares the performance of RMI and DCOM in different scenarios (Roland 2008, p.403).

Scenario one is to remote method with primitive method. In the first scenario, Roland invoke a method formed **void scenario1Method(int data)**, and place this method into a 50000 loop. The result of this experiment is showed below. The result shows that the best choice for TCP is RMI, and .Net is the best choice for HTTP. Roland also do experiment on remoting method with bulk parameter and object parameter in his experiment. The more detail can be found in his experiment report “Java RMI versus .Net Remoting Architectural Comparison and Performance Evaluation”.

Platform	Protocol	Mean value (sec)	Standard Deviation(sec)
RMI	TCP	11,499	0,106
	HTTP	335,784	0,667
	manual	11,392	0,008
.NET	TCP	27,518	0,290
	HTTP	55,275	0,083
	Manual	11,035	0,017

Table 2 the performance of RMI and .NET

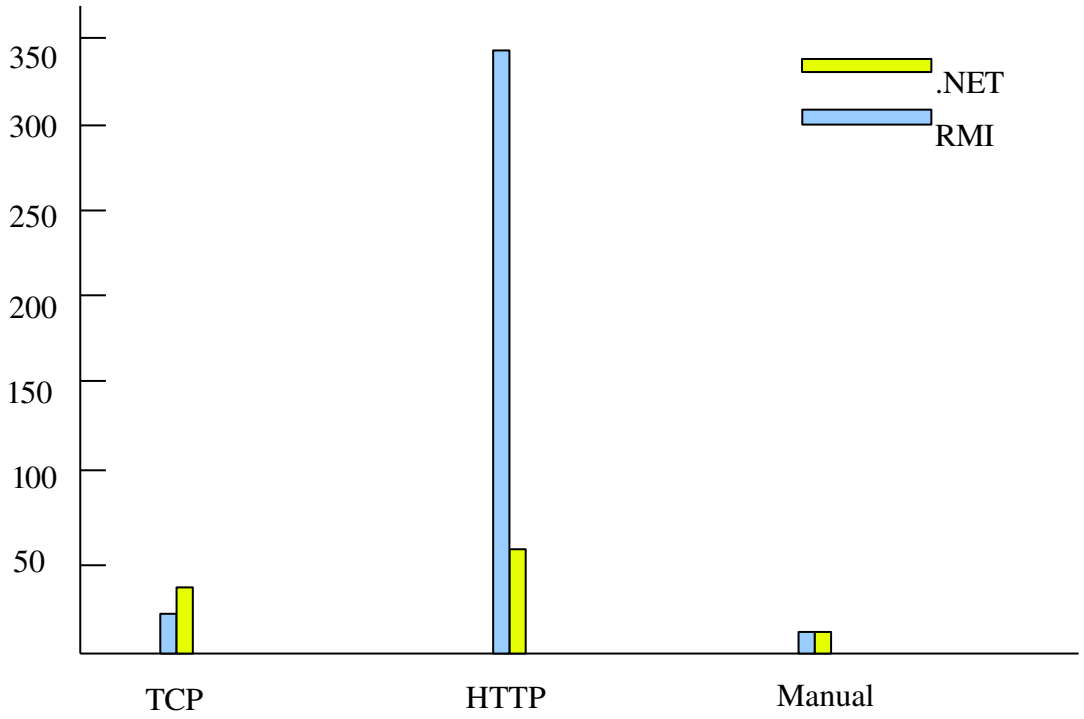


Figure 10 the performance of RMI and .NET

The bellowing table compares some basic concept of these three technologies (Henry, 2005).

	CORBA	**RMI**	**DCOM**
Object Implementation	The IDL need to map into those languages which	The remote object should implement java interact	In DCOM, the object is binary interface, such that is

	include: Ada, Java, C, Lisp, C++, PL_I, COBOL, Python, CORBA Scripting Language, Smalltalk.	Serialization, so only Java support.	programming language independent. Most of language can be used.
Client/Server Interface	stub/skeleton	stub/skeleton	proxy/stub
Remote Protocol	Internet Inter-ORB Protocol (IIOP)	Java Remote Method Protocol (JRMP)	Object Remote Procedure Call (ORPC)
Object Identification	Use Object references as its identify.	ObjID is an identification of the remote server object	CLSID is associated with an object
Object Location and Activation	The ORB is used to locate an object, and Object Adapter is used for activation.	Base on Java Virtual Machine (JVM)	Service Control Manager (SCM) is used to lookup can activate the object in the service side.
Inheritance	The interface implement the CORBA Object	The object implement java.rmi.Remote	The object implement IUnknow or IClassFactory
On-demand Activation	A client binds with a naming service.	Use lookup () and URI to get the find out the ojbect.	CoCreateInstance is used to create a remote object.

Table 3 **CORBA, RMI, and DCOM comparison table**

Andrew compares the service between CORBA and DCOM (Andrew, 2002, p.531).

CORBA Service	DCOM Service
Collection	ActiveX Data Objects
Query	None
Concurrency	COM+ Automatic Transactions
Transaction	COM+ Events
Event	COM+ Events
Notification	Marshalling utilities
Externalization	Marshalling utilities
Life cycle	Class factories, JIT activation
Licensing	Special class factories
Naming	Monikers
Property	None
Trading	None
Persistence	Structured storage
Relationship	None
Security	Authorization
Time	None

Table 4 the comparison of CORBA service and DCOM service

Due to the complex of software development in the current IT industry, the architecture of HTTP/CGI can not meet the requirement of software development. So CORBA and Java are going to merge together, because of these two technologies can remedy the disadvantage of each other. Most of technology companies try to implement Java RMI on top of CORBA IIOP except Microsoft. Java is the best choice for developing a client/server CORBA object. This is because of the native of Java language, such as platform independent, multi thread, exception mechanism and garbage collection. RMI/IIOP enables the developer write the code in Java programming language. The rmic can generate the code for distributed application. The distributed application can interoperate and communicate with each other on IIOP layer. Because of CORBA IDL compiler can map many language, the distributed system which use RMI/IIOP technology can support the any CORBA compliant language, not just pure java. The specification of RMI/IIOP can be found on the Sun official webpage http://java.sun.com/javase/6/docs/technotes/guides/rmi-iiop/ . In Robert's book "Client/Server Programming with Java and CORBA", Robert describes more detail of the benefit of RMI/IIOP, and how to develop RMI/IIOP language.

Conclusion

Because of the development of software is more complex than the earlier day, the distributed object oriented technology becomes more and more important. The distributed object technology can solve some problem of developing software, such as reusable, heterogeneous.

This paper try to have a bird eye’s look of the architecture of three distribute object technologies, and compare their differences. In the CORBA section, this paper describes the architecture of CORBA and ORB. In the implement section, simply describes the steps to create a CORBA application with Java. In RMI section, this paper explains the architecture of RMI four layers. After that, this paper has a quick look of the architecture of DCOM, and explains the benefit of binary interface. Finally, this paper compares some aspect of CORBA, RMI and DCOM, such as performance, basic concept and service.

Table and Figure References

References

Andrew S. Tanenhaum and Maarten van Steen. *DISTRIBUTED SYSTEMS Principles and Paradigms*. Prentice-Hall Inc., USA, 2002.

Frank E. Redmond III. *DCOM Microsoft Distributed Component Object Model*. IDG Books Worldwide, Inc., CA, 1997.

Guy Eddon and Henry Eddon. *Inside Distributed COM*. Microsoft Press., USA, 1998.

Henry Xiao. Middleware and its Applications Research Survey. http://research.cs.queensu.ca/home/xiao/DS/DSreport.html 2005.

Robert Orfali and Dan Harkey. *Client/Server Programming with JAVA and CORBA*. John Wiley & Sons, Inc., USA, seconde edition, 1998.

Roland Schwarzkopf, Markus Mathes, Steffen Heinzl, Bernd Freisleben and Helmut Dohmann. *Java RMI versus .NET Remoting Architectural Comparison and Performance Evaluation*. 2008.

Roger Sessions. *COM and DCOM: Microsoft's Vision for Distributed Objects*. John Wiley & Sons, Inc, CA, 1998.

Troy Bryan Downing. *Java RMI Remote Method Invocation*. IDG Books Worldwide, Inc., CA, 1998.

Vijay K. Garg. Concurrent and Distributed Computing in Java. John Wiley & Sons, Inc. USA, 2004

Wolfgang Emmerich. *Engineering Distributed Objects*. John Wiley & Sons, Ltd, England, 1999.